

Fondamenti dell'informatica

Corso di Laurea Triennale in Informatica
Prof. Stefano Berardi, Prof. Felice Cardone, Prof. Luca Paolini
Università degli Studi di Torino

Richard Johnsonbaugh

J. Glenn Brookshear
Dennis Brylow



PEARSON.TEXT.BUILDER



computer science

AN OVERVIEW

12th Edition

Global Edition

J. Glenn Brookshear
and
Dennis Brylow

Global Edition contributions by
Manasa S.

PEARSON

Boston Columbus Indianapolis New York San Francisco Upper Saddle River
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montréal Toronto
Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo



Contents

Chapter 0 Introduction 13

- 0.1 The Role of Algorithms 14
- 0.2 The History of Computing 16
- 0.3 An Outline of Our Study 21
- 0.4 The Overarching Themes of Computer Science 23

Chapter 1 Data Storage 31

- 1.1 Bits and Their Storage 32
- 1.2 Main Memory 38
- 1.3 Mass Storage 41
- 1.4 Representing Information as Bit Patterns 46
- *1.5 The Binary System 52
- *1.6 Storing Integers 58
- *1.7 Storing Fractions 64
- *1.8 Data and Programming 69
- *1.9 Data Compression 75
- *1.10 Communication Errors 81

Chapter 2 Data Manipulation 93

- 2.1 Computer Architecture 94
- 2.2 Machine Language 97
- 2.3 Program Execution 103
- *2.4 Arithmetic/Logic Instructions 110
- *2.5 Communicating with Other Devices 115
- *2.6 Programming Data Manipulation 120
- *2.7 Other Architectures 129

Chapter 3 Operating Systems 139

- 3.1 The History of Operating Systems 140
- 3.2 Operating System Architecture 144
- 3.3 Coordinating the Machine's Activities 152

**Asterisks indicate suggestions for optional sections.*

| | |
|---|-----|
| *3.4 Handling Competition Among Processes | 155 |
| 3.5 Security | 160 |

Chapter 4 Networking and the Internet 169

| | |
|--------------------------|-----|
| 4.1 Network Fundamentals | 170 |
| 4.2 The Internet | 179 |
| 4.3 The World Wide Web | 188 |
| *4.4 Internet Protocols | 197 |
| 4.5 Security | 203 |

Chapter 5 Algorithms 217

| | |
|---------------------------------|-----|
| 5.1 The Concept of an Algorithm | 218 |
| 5.2 Algorithm Representation | 221 |
| 5.3 Algorithm Discovery | 228 |
| 5.4 Iterative Structures | 234 |
| 5.5 Recursive Structures | 245 |
| 5.6 Efficiency and Correctness | 253 |

Chapter 6 Programming Languages 271

| | |
|--|-----|
| 6.1 Historical Perspective | 272 |
| 6.2 Traditional Programming Concepts | 280 |
| 6.3 Procedural Units | 292 |
| 6.4 Language Implementation | 300 |
| 6.5 Object-Oriented Programming | 308 |
| *6.6 Programming Concurrent Activities | 315 |
| *6.7 Declarative Programming | 318 |

Chapter 7 Software Engineering 331

| | |
|---|-----|
| 7.1 The Software Engineering Discipline | 332 |
| 7.2 The Software Life Cycle | 334 |
| 7.3 Software Engineering Methodologies | 338 |
| 7.4 Modularity | 341 |
| 7.5 Tools of the Trade | 348 |
| 7.6 Quality Assurance | 356 |
| 7.7 Documentation | 360 |
| 7.8 The Human-Machine Interface | 361 |
| 7.9 Software Ownership and Liability | 364 |

Chapter 8 Data Abstractions 373

| | |
|-----------------------------------|-----|
| 8.1 Basic Data Structures | 374 |
| 8.2 Related Concepts | 377 |
| 8.3 Implementing Data Structures | 380 |
| 8.4 A Short Case Study | 394 |
| 8.5 Customized Data Types | 399 |
| 8.6 Classes and Objects | 403 |
| *8.7 Pointers in Machine Language | 405 |

Chapter 9 Database Systems 415

- 9.1 Database Fundamentals 416
- 9.2 The Relational Model 421
- *9.3 Object-Oriented Databases 432
- *9.4 Maintaining Database Integrity 434
- *9.5 Traditional File Structures 438
- 9.6 Data Mining 446
- 9.7 Social Impact of Database Technology 448

Chapter 10 Computer Graphics 457

- 10.1 The Scope of Computer Graphics 458
- 10.2 Overview of 3D Graphics 460
- 10.3 Modeling 461
- 10.4 Rendering 469
- *10.5 Dealing with Global Lighting 480
- 10.6 Animation 483

Chapter 11 Artificial Intelligence 491

- 11.1 Intelligence and Machines 492
- 11.2 Perception 497
- 11.3 Reasoning 503
- 11.4 Additional Areas of Research 514
- 11.5 Artificial Neural Networks 519
- 11.6 Robotics 526
- 11.7 Considering the Consequences 529

Chapter 12 Theory of Computation 539

- 12.1 Functions and Their Computation 540
- 12.2 Turing Machines 542
- 12.3 Universal Programming Languages 546
- 12.4 A Noncomputable Function 552
- 12.5 Complexity of Problems 556
- *12.6 Public-Key Cryptography 565

Appendices 575

- A ASCII 577
- B Circuits to Manipulate Two's Complement Representations 578
- C A Simple Machine Language 581
- D High-Level Programming Languages 583
- E The Equivalence of Iterative and Recursive Structures 585
- F Answers to Questions & Exercises 587

Index 629

Introduction

In this preliminary chapter we consider the scope of computer science, develop a historical perspective, and establish a foundation from which to launch our study.

0.1 The Role of Algorithms

0.2 The History of Computing

0.3 An Outline of Our Study

0.4 The Overarching Themes of Computer Science

Algorithms
Abstraction
Creativity

Data
Programming
Internet
Impact

Computer science is the discipline that seeks to build a scientific foundation for such topics as computer design, computer programming, information processing, algorithmic solutions of problems, and the algorithmic process itself. It provides the underpinnings for today's computer applications as well as the foundations for tomorrow's computing infrastructure.

This book provides a comprehensive introduction to this science. We will investigate a wide range of topics including most of those that constitute a typical university computer science curriculum. We want to appreciate the full scope and dynamics of the field. Thus, in addition to the topics themselves, we will be interested in their historical development, the current state of research, and prospects for the future. Our goal is to establish a functional understanding of computer science—one that will support those who wish to pursue more specialized studies in the science as well as one that will enable those in other fields to flourish in an increasingly technical society.

0.1 The Role of Algorithms

We begin with the most fundamental concept of computer science—that of an **algorithm**. Informally, an algorithm is a set of steps that defines how a task is performed. (We will be more precise later in Chapter 5.) For example, there are algorithms for cooking (called recipes), for finding your way through a strange city (more commonly called directions), for operating washing machines (usually displayed on the inside of the washer's lid or perhaps on the wall of a laundromat), for playing music (expressed in the form of sheet music), and for performing magic tricks (Figure 0.1).

Before a machine such as a computer can perform a task, an algorithm for performing that task must be discovered and represented in a form that is compatible with the machine. A representation of an algorithm is called a **program**. For the convenience of humans, computer programs are usually printed on paper or displayed on computer screens. For the convenience of machines, programs are encoded in a manner compatible with the technology of the machine. The process of developing a program, encoding it in machine-compatible form, and inserting it into a machine is called **programming**. Programs, and the algorithms they represent, are collectively referred to as **software**, in contrast to the machinery itself, which is known as **hardware**.

The study of algorithms began as a subject in mathematics. Indeed, the search for algorithms was a significant activity of mathematicians long before the development of today's computers. The goal was to find a single set of directions that described how all problems of a particular type could be solved. One of the best known examples of this early research is the long division algorithm for finding the quotient of two multiple-digit numbers. Another example is the Euclidean algorithm, discovered by the ancient Greek mathematician Euclid, for finding the greatest common divisor of two positive integers (Figure 0.2).

Once an algorithm for performing a task has been found, the performance of that task no longer requires an understanding of the principles on which the algorithm is based. Instead, the performance of the task is reduced to the process of merely following directions. (We can follow the long division algorithm to find a quotient or the Euclidean algorithm to find a greatest common divisor without

Figure 0.1 An algorithm for a magic trick

Effect: The performer places some cards from a normal deck of playing cards face down on a table and mixes them thoroughly while spreading them out on the table. Then, as the audience requests either red or black cards, the performer turns over cards of the requested color.

Secret and Patter:

- Step 1. From a normal deck of cards, select ten red cards and ten black cards. Deal these cards face up in two piles on the table according to color.
- Step 2. Announce that you have selected some red cards and some black cards.
- Step 3. Pick up the red cards. Under the pretense of aligning them into a small deck, hold them face down in your left hand and, with the thumb and first finger of your right hand, pull back on each end of the deck so that each card is given a slightly *backward* curve. Then place the deck of red cards face down on the table as you say, "Here are the red cards in this stack."
- Step 4. Pick up the black cards. In a manner similar to that in step 3, give these cards a slight *forward* curve. Then return these cards to the table in a face-down deck as you say, "And here are the black cards in this stack."
- Step 5. Immediately after returning the black cards to the table, use both hands to mix the red and black cards (still face down) as you spread them out on the tabletop. Explain that you are thoroughly mixing the cards.
- Step 6. As long as there are face-down cards on the table, repeatedly execute the following steps:
 - 6.1. Ask the audience to request either a red or a black card.
 - 6.2. If the color requested is red and there is a face-down card with a concave appearance, turn over such a card while saying, "Here is a red card."
 - 6.3. If the color requested is black and there is a face-down card with a convex appearance, turn over such a card while saying, "Here is a black card."
 - 6.4. Otherwise, state that there are no more cards of the requested color and turn over the remaining cards to prove your claim.

Figure 0.2 The Euclidean algorithm for finding the greatest common divisor of two positive integers

Description: This algorithm assumes that its input consists of two positive integers and proceeds to compute the greatest common divisor of these two values.

Procedure:

- Step 1. Assign M and N the value of the larger and smaller of the two input values, respectively.
- Step 2. Divide M by N, and call the remainder R.
- Step 3. If R is not 0, then assign M the value of N, assign N the value of R, and return to step 2; otherwise, the greatest common divisor is the value currently assigned to N.

understanding why the algorithm works.) In a sense, the intelligence required to solve the problem at hand is encoded in the algorithm.

Capturing and conveying intelligence (or at least intelligent behavior) by means of algorithms allows us to build machines that perform useful tasks. Consequently, the level of intelligence displayed by machines is limited by the intelligence that can be conveyed through algorithms. We can construct a machine to perform a task only if an algorithm exists for performing that task. In turn, if no algorithm exists for solving a problem, then the solution of that problem lies beyond the capabilities of machines.

Identifying the limitations of algorithmic capabilities solidified as a subject in mathematics in the 1930s with the publication of Kurt Gödel's incompleteness theorem. This theorem essentially states that in any mathematical theory encompassing our traditional arithmetic system, there are statements whose truth or falseness cannot be established by algorithmic means. In short, any complete study of our arithmetic system lies beyond the capabilities of algorithmic activities. This realization shook the foundations of mathematics, and the study of algorithmic capabilities that ensued was the beginning of the field known today as computer science. Indeed, it is the study of algorithms that forms the core of computer science.

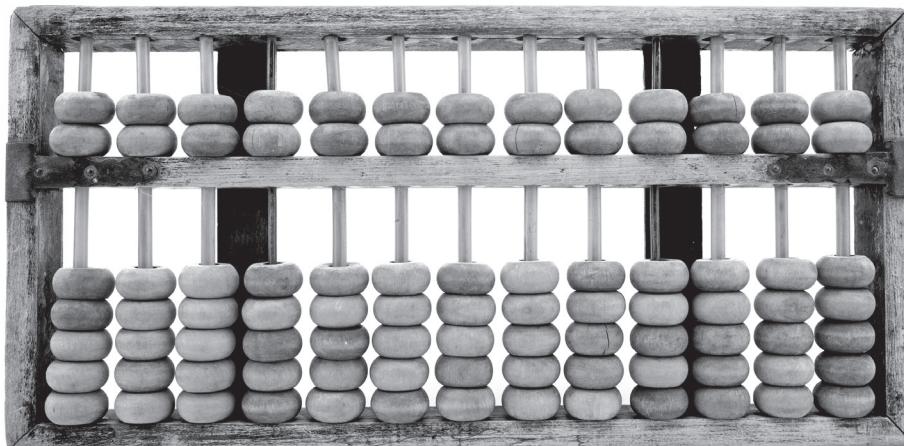
0.2 The History of Computing

Today's computers have an extensive genealogy. One of the earlier computing devices was the abacus. History tells us that it probably had its roots in ancient China and was used in the early Greek and Roman civilizations. The machine is quite simple, consisting of beads strung on rods that are in turn mounted in a rectangular frame (Figure 0.3). As the beads are moved back and forth on the rods, their positions represent stored values. It is in the positions of the beads that this "computer" represents and stores data. For control of an algorithm's execution, the machine relies on the human operator. Thus the abacus alone is merely a data storage system; it must be combined with a human to create a complete computational machine.

In the time period after the Middle Ages and before the Modern Era, the quest for more sophisticated computing machines was seeded. A few inventors began to experiment with the technology of gears. Among these were Blaise Pascal (1623–1662) of France, Gottfried Wilhelm Leibniz (1646–1716) of Germany, and Charles Babbage (1792–1871) of England. These machines represented data through gear positioning, with data being entered mechanically by establishing initial gear positions. Output from Pascal's and Leibniz's machines was achieved by observing the final gear positions. Babbage, on the other hand, envisioned machines that would print results of computations on paper so that the possibility of transcription errors would be eliminated.

As for the ability to follow an algorithm, we can see a progression of flexibility in these machines. Pascal's machine was built to perform only addition. Consequently, the appropriate sequence of steps was embedded into the structure of the machine itself. In a similar manner, Leibniz's machine had its algorithms firmly embedded in its architecture, although the operator could select from a variety of arithmetic operations it offered. Babbage's Difference Engine

Figure 0.3 Chinese wooden abacus (Pink Badger/Fotolia)

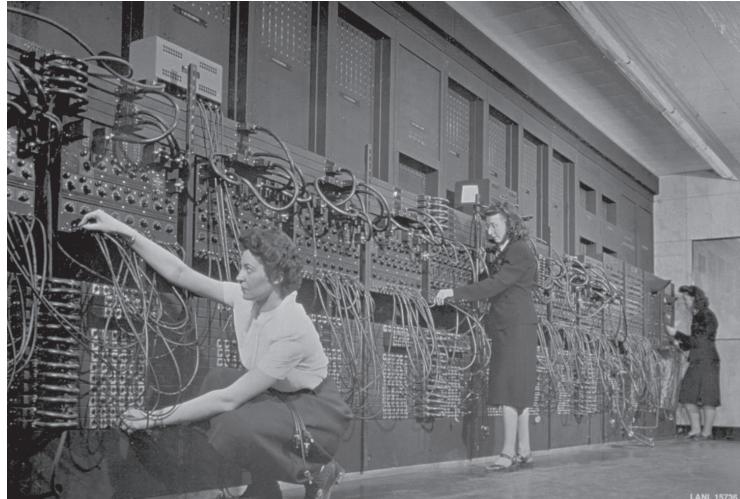


(of which only a demonstration model was constructed) could be modified to perform a variety of calculations, but his Analytical Engine (never funded for construction) was designed to read instructions in the form of holes in paper cards. Thus Babbage's Analytical Engine was programmable. In fact, Augusta Ada Byron (Ada Lovelace), who published a paper in which she demonstrated how Babbage's Analytical Engine could be programmed to perform various computations, is often identified today as the world's first programmer.

The idea of communicating an algorithm via holes in paper was not originated by Babbage. He got the idea from Joseph Jacquard (1752–1834), who, in 1801, had developed a weaving loom in which the steps to be performed during the weaving process were determined by patterns of holes in large thick cards made of wood (or cardboard). In this manner, the algorithm followed by the loom could be changed easily to produce different woven designs. Another beneficiary of Jacquard's idea was Herman Hollerith (1860–1929), who applied the concept of representing information as holes in paper cards to speed up the tabulation process in the 1890 U.S. census. (It was this work by Hollerith that led to the creation of IBM.) Such cards ultimately came to be known as punched cards and survived as a popular means of communicating with computers well into the 1970s.

Nineteenth-century technology was unable to produce the complex gear-driven machines of Pascal, Leibniz, and Babbage cost-effectively. But with the advances in electronics in the early 1900s, this barrier was overcome. Examples of this progress include the electromechanical machine of George Stibitz, completed in 1940 at Bell Laboratories, and the Mark I, completed in 1944 at Harvard University by Howard Aiken and a group of IBM engineers. These machines made heavy use of electronically controlled mechanical relays. In this sense they were obsolete almost as soon as they were built, because other researchers were applying the technology of vacuum tubes to construct totally

Figure 0.4 Three women operating the ENIAC's (Electronic Numerical Integrator And Computer) main control panel while the machine was at the Moore School. The machine was later moved to the U.S. Army's Ballistics Research Laboratory. (Courtesy U.S. Army.)



electronic computers. The first of these vacuum tube machines was apparently the Atanasoff-Berry machine, constructed during the period from 1937 to 1941 at Iowa State College (now Iowa State University) by John Atanasoff and his assistant, Clifford Berry. Another was a machine called Colossus, built under the direction of Tommy Flowers in England to decode German messages during the latter part of World War II. (Actually, as many as ten of these machines were apparently built, but military secrecy and issues of national security kept their existence from becoming part of the “computer family tree.”) Other, more flexible machines, such as the ENIAC (electronic numerical integrator and calculator) developed by John Mauchly and J. Presper Eckert at the Moore School of Electrical Engineering, University of Pennsylvania, soon followed (Figure 0.4).

From that point on, the history of computing machines has been closely linked to advancing technology, including the invention of transistors (for which physicists William Shockley, John Bardeen, and Walter Brattain were awarded a Nobel Prize) and the subsequent development of complete circuits constructed as single units, called integrated circuits (for which Jack Kilby also won a Nobel Prize in physics). With these developments, the room-sized machines of the 1940s were reduced over the decades to the size of single cabinets. At the same time, the processing power of computing machines began to double every two years (a trend that has continued to this day). As work on integrated circuitry progressed, many of the components within a computer became readily available on the open market as integrated circuits encased in toy-sized blocks of plastic called chips.

A major step toward popularizing computing was the development of desktop computers. The origins of these machines can be traced to the computer hobbyists who built homemade computers from combinations of chips. It was within this “underground” of hobby activity that Steve Jobs and Stephen Wozniak

Babbage's Difference Engine

The machines designed by Charles Babbage were truly the forerunners of modern computer design. If technology had been able to produce his machines in an economically feasible manner and if the data processing demands of commerce and government had been on the scale of today's requirements, Babbage's ideas could have led to a computer revolution in the 1800s. As it was, only a demonstration model of his Difference Engine was constructed in his lifetime. This machine determined numerical values by computing "successive differences." We can gain an insight to this technique by considering the problem of computing the squares of the integers. We begin with the knowledge that the square of 0 is 0, the square of 1 is 1, the square of 2 is 4, and the square of 3 is 9. With this, we can determine the square of 4 in the following manner (see the following diagram). We first compute the differences of the squares we already know: $1^2 - 0^2 = 1$, $2^2 - 1^2 = 3$, and $3^2 - 2^2 = 5$. Then we compute the differences of these results: $3 - 1 = 2$, and $5 - 3 = 2$. Note that these differences are both 2. Assuming that this consistency continues (mathematics can show that it does), we conclude that the difference between the value ($4^2 - 3^2$) and the value ($3^2 - 2^2$) must also be 2. Hence $(4^2 - 3^2)$ must be 2 greater than $(3^2 - 2^2)$, so $4^2 - 3^2 = 7$ and thus $4^2 = 3^2 + 7 = 16$. Now that we know the square of 4, we could continue our procedure to compute the square of 5 based on the values of 1^2 , 2^2 , 3^2 , and 4^2 . (Although a more in-depth discussion of successive differences is beyond the scope of our current study, students of calculus may wish to observe that the preceding example is based on the fact that the derivative of $y = x^2$ is a straight line with a slope of 2.)

| x | x^2 | First difference | Second difference |
|-----|-------|------------------|-------------------|
| 0 | 0 | | |
| 1 | 1 | 1 | |
| 2 | 4 | 3 | 2 |
| 3 | 9 | 5 | |
| 4 | 16 | 7 | 2 |
| 5 | | | 2 |

built a commercially viable home computer and, in 1976, established Apple Computer, Inc. (now Apple Inc.) to manufacture and market their products. Other companies that marketed similar products were Commodore, Heathkit, and Radio Shack. Although these products were popular among computer hobbyists, they were not widely accepted by the business community, which continued to look to the well-established IBM and its large mainframe computers for the majority of its computing needs.

In 1981, IBM introduced its first desktop computer, called the personal computer, or PC, whose underlying software was developed by a newly formed company known as Microsoft. The PC was an instant success and legitimized

Augusta Ada Byron

Augusta Ada Byron, Countess of Lovelace, has been the subject of much commentary in the computing community. She lived a somewhat tragic life of less than 37 years (1815–1852) that was complicated by poor health and the fact that she was a non-conformist in a society that limited the professional role of women. Although she was interested in a wide range of science, she concentrated her studies in mathematics. Her interest in “compute science” began when she became fascinated by the machines of Charles Babbage at a demonstration of a prototype of his Difference Engine in 1833. Her contribution to computer science stems from her translation from French into English of a paper discussing Babbage’s designs for the Analytical Engine. To this translation, Babbage encouraged her to attach an addendum describing applications of the engine and containing examples of how the engine could be programmed to perform various tasks. Babbage’s enthusiasm for Ada Byron’s work was apparently motivated by his hope that its publication would lead to financial backing for the construction of his Analytical Engine. (As the daughter of Lord Byron, Ada Byron held celebrity status with potentially significant financial connections.) This backing never materialized, but Ada Byron’s addendum has survived and is considered to contain the first examples of computer programs. The degree to which Babbage influenced Ada Byron’s work is debated by historians. Some argue that Babbage made major contributions, whereas others contend that he was more of an obstacle than an aid. Nonetheless, Augusta Ada Byron is recognized today as the world’s first programmer, a status that was certified by the U.S. Department of Defense when it named a prominent programming language (Ada) in her honor.

the desktop computer as an established commodity in the minds of the business community. Today, the term PC is widely used to refer to all those machines (from various manufacturers) whose design has evolved from IBM’s initial desktop computer, most of which continue to be marketed with software from Microsoft. At times, however, the term PC is used interchangeably with the generic terms *desktop* or *laptop*.

As the twentieth century drew to a close, the ability to connect individual computers in a world-wide system called the **Internet** was revolutionizing communication. In this context, Tim Berners-Lee (a British scientist) proposed a system by which documents stored on computers throughout the Internet could be linked together producing a maze of linked information called the **World Wide Web** (often shortened to “Web”). To make the information on the Web accessible, software systems, called **search engines**, were developed to “sift through” the Web, “categorize” their findings, and then use the results to assist users researching particular topics. Major players in this field are Google, Yahoo, and Microsoft. These companies continue to expand their Web-related activities, often in directions that challenge our traditional way of thinking.

At the same time that desktop and laptop computers were being accepted and used in homes, the miniaturization of computing machines continued. Today, tiny computers are embedded within a wide variety of electronic appliances and devices. Automobiles may now contain dozens of small computers running Global Positioning Systems (GPS), monitoring the function of the engine, and providing

Google

Founded in 1998, Google Inc. has become one of the world's most recognized technology companies. Its core service, the Google search engine, is used by millions of people to find documents on the World Wide Web. In addition, Google provides electronic mail service (called Gmail), an Internet-based video-sharing service (called YouTube), and a host of other Internet services (including Google Maps, Google Calendar, Google Earth, Google Books, and Google Translate).

However, in addition to being a prime example of the entrepreneurial spirit, Google also provides examples of how expanding technology is challenging society. For example, Google's search engine has led to questions regarding the extent to which an international company should comply with the wishes of individual governments; YouTube has raised questions regarding the extent to which a company should be liable for information that others distribute through its services as well as the degree to which the company can claim ownership of that information; Google Books has generated concerns regarding the scope and limitations of intellectual property rights; and Google Maps has been accused of violating privacy rights.

voice command services for controlling the car's audio and phone communication systems.

Perhaps the most revolutionary application of computer miniaturization is found in the expanding capabilities of **smartphones**, hand-held general-purpose computers on which telephony is only one of many applications. More powerful than the supercomputers of prior decades, these pocket-sized devices are equipped with a rich array of sensors and interfaces including cameras, microphones, compasses, touch screens, accelerometers (to detect the phone's orientation and motion), and a number of wireless technologies to communicate with other smartphones and computers. Many argue that the smartphone is having a greater effect on global society than the PC revolution.

0.3 An Outline of Our Study

This text follows a bottom-up approach to the study of computer science, beginning with such hands-on topics as computer hardware and leading to the more abstract topics such as algorithm complexity and computability. The result is that our study follows a pattern of building larger and larger abstract tools as our understanding of the subject expands.

We begin by considering topics dealing with the design and construction of machines for executing algorithms. In Chapter 1 (Data Storage), we look at how information is encoded and stored within modern computers, and in Chapter 2 (Data Manipulation), we investigate the basic internal operation of a simple computer. Although part of this study involves technology, the general theme is technology independent. That is, such topics as digital circuit design, data encoding and compression systems, and computer architecture are relevant over a wide range of technology and promise to remain relevant regardless of the direction of future technology.

In Chapter 3 (Operating Systems), we study the software that controls the overall operation of a computer. This software is called an operating system. It is a computer's operating system that controls the interface between the machine and its outside world, protecting the machine and the data stored within from unauthorized access, allowing a computer user to request the execution of various programs, and coordinating the internal activities required to fulfill the user's requests.

In Chapter 4 (Networking and the Internet), we study how computers are connected to each other to form computer networks and how networks are connected to form internets. This study leads to topics such as network protocols, the Internet's structure and internal operation, the World Wide Web, and numerous issues of security.

Chapter 5 (Algorithms) introduces the study of algorithms from a more formal perspective. We investigate how algorithms are discovered, identify several fundamental algorithmic structures, develop elementary techniques for representing algorithms, and introduce the subjects of algorithm efficiency and correctness.

In Chapter 6 (Programming Languages), we consider the subject of algorithm representation and the program development process. Here we find that the search for better programming techniques has led to a variety of programming methodologies or paradigms, each with its own set of programming languages. We investigate these paradigms and languages as well as consider issues of grammar and language translation.

Chapter 7 (Software Engineering) introduces the branch of computer science known as software engineering, which deals with the problems encountered when developing large software systems. The underlying theme is that the design of large software systems is a complex task that embraces problems beyond those of traditional engineering. Thus, the subject of software engineering has become an important field of research within computer science, drawing from such diverse fields as engineering, project management, personnel management, programming language design, and even architecture.

In the next two chapters we look at ways data can be organized within a computer system. In Chapter 8 (Data Abstractions), we introduce techniques traditionally used for organizing data in a computer's main memory and then trace the evolution of data abstraction from the concept of primitives to today's object-oriented techniques. In Chapter 9 (Database Systems), we consider methods traditionally used for organizing data in a computer's mass storage and investigate how extremely large and complex database systems are implemented.

In Chapter 10 (Computer Graphics), we explore the subject of graphics and animation, a field that deals with creating and photographing virtual worlds. Based on advancements in the more traditional areas of computer science such as machine architecture, algorithm design, data structures, and software engineering, the discipline of graphics and animation has seen significant progress and has now blossomed into an exciting, dynamic subject. Moreover, the field exemplifies how various components of computer science combine with other disciplines such as physics, art, and photography to produce striking results.

In Chapter 11 (Artificial Intelligence), we learn that to develop more useful machines computer science has turned to the study of human intelligence for insight. The hope is that by understanding how our own minds reason and

perceive, researchers will be able to design algorithms that mimic these processes and thus transfer comparable capabilities to machines. The result is the area of computer science known as artificial intelligence, which leans heavily on research in such areas as psychology, biology, and linguistics.

We close our study with Chapter 12 (Theory of Computation) by investigating the theoretical foundations of computer science—a subject that allows us to understand the limitations of algorithms (and thus machines). Here we identify some problems that cannot be solved algorithmically (and therefore lie beyond the capabilities of machines) as well as learn that the solutions to many other problems require such enormous time or space that they are also unsolvable from a practical perspective. Thus, it is through this study that we are able to grasp the scope and limitations of algorithmic systems.

In each chapter, our goal is to explore the subject deeply enough to enable true understanding. We want to develop a working knowledge of computer science—a knowledge that will allow you to understand the technical society in which you live and to provide a foundation from which you can learn as science and technology advance.

0.4 The Overarching Themes of Computer Science

In addition to the main topics of each chapter as listed above, we also hope to broaden your understanding of computer science by incorporating several overarching themes.

The miniaturization of computers and their expanding capabilities have brought computer technology to the forefront of today's society, and computer technology is so prevalent that familiarity with it is fundamental to being a member of the modern world. Computing technology has altered the ability of governments to exert control; had enormous impact on global economics; led to startling advances in scientific research; revolutionized the role of data collection, storage, and applications; provided new means for people to communicate and interact; and has repeatedly challenged society's status quo. The result is a proliferation of subjects surrounding computer science, each of which is now a significant field of study in its own right. Moreover, as with mechanical engineering and physics, it is often difficult to draw a line between these fields and computer science itself. Thus, to gain a proper perspective, our study will not only cover topics central to the core of computer science but also will explore a variety of disciplines dealing with both applications and consequences of the science. Indeed, an introduction to computer science is an interdisciplinary undertaking.

As we set out to explore the breadth of the field of computing, it is helpful to keep in mind the main themes that unite computer science. While the codification of the "Seven Big Ideas of Computer Science"¹ postdates the first ten editions of this book, they closely parallel the themes of the chapters to come. The "Seven Big Ideas" are, briefly: Algorithms, Abstraction, Creativity, Data, Programming, Internet, and Impact. In the chapters that follow, we include a variety of topics, in each case introducing central ideas of the topic, current areas of research, and some of the techniques being applied to advance knowledge in that realm. Watch for the "Big Ideas" as we return to them again and again.

¹www.csprinciples.org

Algorithms

Limited data storage capabilities and intricate, time-consuming programming procedures restricted the complexity of the algorithms used in the earliest computing machines. However, as these limitations began to disappear, machines were applied to increasingly larger and more complex tasks. As attempts to express the composition of these tasks in algorithmic form began to tax the abilities of the human mind, more and more research efforts were directed toward the study of algorithms and the programming process.

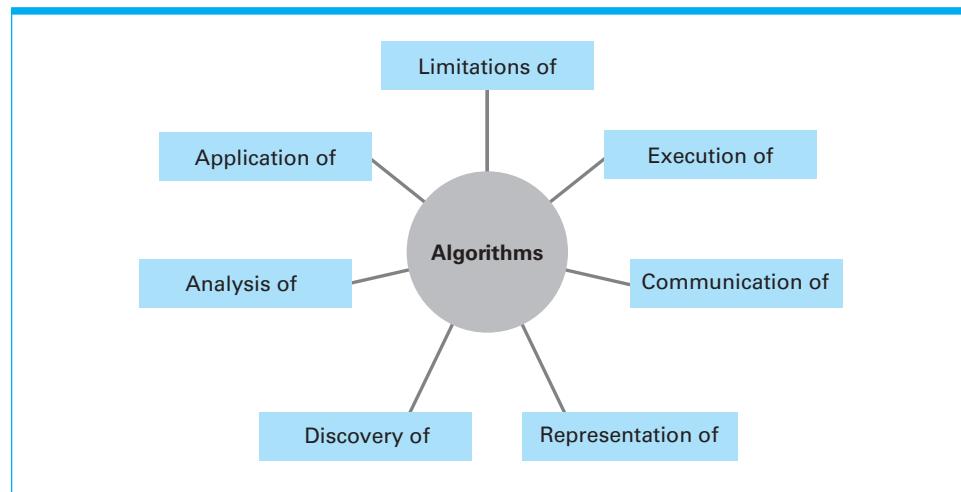
It was in this context that the theoretical work of mathematicians began to pay dividends. As a consequence of Gödel's incompleteness theorem, mathematicians had already been investigating those questions regarding algorithmic processes that advancing technology was now raising. With that, the stage was set for the emergence of a new discipline known as *computer science*.

Today, computer science has established itself as the science of algorithms. The scope of this science is broad, drawing from such diverse subjects as mathematics, engineering, psychology, biology, business administration, and linguistics. Indeed, researchers in different branches of computer science may have very distinct definitions of the science. For example, a researcher in the field of computer architecture may focus on the task of miniaturizing circuitry and thus view computer science as the advancement and application of technology. But, a researcher in the field of database systems may see computer science as seeking ways to make information systems more useful. And, a researcher in the field of artificial intelligence may regard computer science as the study of intelligence and intelligent behavior.

Nevertheless, all of these researchers are involved in aspects of the science of algorithms. Given the central role that algorithms play in computer science (see Figure 0.5), it is instructive to identify some questions that will provide focus for our study of this big idea.

- Which problems can be solved by algorithmic processes?
- How can the discovery of algorithms be made easier?

Figure 0.5 The central role of algorithms in computer science



- How can the techniques of representing and communicating algorithms be improved?
- How can the characteristics of different algorithms be analyzed and compared?
- How can algorithms be used to manipulate information?
- How can algorithms be applied to produce intelligent behavior?
- How does the application of algorithms affect society?

Abstraction

The term **abstraction**, as we are using it here, refers to the distinction between the external properties of an entity and the details of the entity's internal composition. It is abstraction that allows us to ignore the internal details of a complex device such as a computer, automobile, or microwave oven and use it as a single, comprehensible unit. Moreover, it is by means of abstraction that such complex systems are designed and manufactured in the first place. Computers, automobiles, and microwave ovens are constructed from components, each of which represents a level of abstraction at which the use of the component is isolated from the details of the component's internal composition.

It is by applying abstraction that we are able to construct, analyze, and manage large, complex computer systems that would be overwhelming if viewed in their entirety at a detailed level. At each level of abstraction, we view the system in terms of components, called **abstract tools**, whose internal composition we ignore. This allows us to concentrate on how each component interacts with other components at the same level and how the collection as a whole forms a higher-level component. Thus we are able to comprehend the part of the system that is relevant to the task at hand rather than being lost in a sea of details.

We emphasize that abstraction is not limited to science and technology. It is an important simplification technique with which our society has created a lifestyle that would otherwise be impossible. Few of us understand how the various conveniences of daily life are actually implemented. We eat food and wear clothes that we cannot produce by ourselves. We use electrical devices and communication systems without understanding the underlying technology. We use the services of others without knowing the details of their professions. With each new advancement, a small part of society chooses to specialize in its implementation, while the rest of us learn to use the results as abstract tools. In this manner, society's warehouse of abstract tools expands, and society's ability to progress increases.

Abstraction is a recurring pillar of our study. We will learn that computing equipment is constructed in levels of abstract tools. We will also see that the development of large software systems is accomplished in a modular fashion in which each module is used as an abstract tool in larger modules. Moreover, abstraction plays an important role in the task of advancing computer science itself, allowing researchers to focus attention on particular areas within a complex field. In fact, the organization of this text reflects this characteristic of the science. Each chapter, which focuses on a particular area within the science, is often surprisingly independent of the others, yet together the chapters form a comprehensive overview of a vast field of study.

Creativity

While computers may merely be complex machines mechanically executing rote algorithmic instructions, we shall see that the field of computer science is an inherently creative one. Discovering and applying new algorithms is a human activity that depends on our innate desire to apply our tools to solve problems in the world around us. Computer science not only extends forms of expression spanning the visual, language and musical arts, but also enables new modes of digital expression that pervade the modern world.

Creating large software systems is much less like following a cookbook recipe than it is like conceiving of a grand new sculpture. Envisioning its form and function requires careful planning. Fabricating its components requires time, attention to detail, and practiced skill. The final product embodies the design aesthetics and sensibilities of its creators.

Data

Computers are capable of representing any information that can be discretized and digitized. Algorithms can process or transform such digitally represented information in a dizzying variety of ways. The result of this is not merely the shuffling of digital data from one part of the computer to another; computer algorithms enable us to search for patterns, to create simulations, and to correlate connections in ways that generate new knowledge and insight. Massive storage capacities, high-speed computer networks, and powerful computational tools are driving discoveries in many other disciplines of science, engineering and the humanities. Whether predicting the effects of a new drug by simulating complex protein folding, statistically analyzing the evolution of language across centuries of digitized books, or rendering 3D images of internal organs from a noninvasive medical scan, data is driving modern discovery across the breadth of human endeavors.

Some of the questions about data that we will explore in our study include:

- How do computers store data about common digital artifacts, such as numbers, text, images, sounds, and video?
- How do computers approximate data about analog artifacts in the real world?
- How do computers detect and prevent errors in data?
- What are the ramifications of an ever-growing and interconnected digital universe of data at our disposal?

Programming

Translating human intentions into executable computer algorithms is now broadly referred to as *programming*, although the proliferation of languages and tools available now bear little resemblance to the programmable computers of the 1950s and early 1960s. While computer science consists of much more than computer programming, the ability to solve problems by devising executable algorithms (programs) remains a foundational skill for all computer scientists.

Computer hardware is capable of executing only relatively simple algorithmic steps, but the abstractions provided by computer programming languages allow

humans to reason about and encode solutions for far more complex problems. Several key questions will frame our discussion of this theme.

- How are programs built?
- What kinds of errors can occur in programs?
- How are errors in programs found and repaired?
- What are the effects of errors in modern programs?
- How are programs documented and evaluated?

Internet

The Internet connects computers and electronic devices around the world and has had a profound impact in the way that our technological society stores, retrieves, and shares information. Commerce, news, entertainment, and communication now depend increasingly on this interconnected web of smaller computer networks. Our discussion will not only describe the mechanisms of the Internet as an artifact, but will also touch on the many aspects of human society that are now intertwined with the global network.

The reach of the Internet also has profound implications for our privacy and the security of our personal information. Cyberspace harbors many dangers. Consequently, cryptography and cybersecurity are of growing importance in our connected world.

Impact

Computer science not only has profound impacts on the technologies we use to communicate, work, and play, it also has enormous social repercussions. Progress in computer science is blurring many distinctions on which our society has based decisions in the past and is challenging many of society's long-held principles. In law, it generates questions regarding the degree to which intellectual property can be owned and the rights and liabilities that accompany that ownership. In ethics, it generates numerous options that challenge the traditional principles on which social behavior is based. In government, it generates debates regarding the extent to which computer technology and its applications should be regulated. In philosophy, it generates contention between the presence of intelligent behavior and the presence of intelligence itself. And, throughout society, it generates disputes concerning whether new applications represent new freedoms or new controls.

Such topics are important for those contemplating careers in computing or computer-related fields. Revelations within science have sometimes found controversial applications, causing serious discontent for the researchers involved. Moreover, an otherwise successful career can quickly be derailed by an ethical misstep.

The ability to deal with the dilemmas posed by advancing computer technology is also important for those outside its immediate realm. Indeed, technology is infiltrating society so rapidly that few, if any, are independent of its effects.

This text provides the technical background needed to approach the dilemmas generated by computer science in a rational manner. However, technical knowledge of the science alone does not provide solutions to all the questions involved. With this in mind, this text includes several sections that are devoted to social, ethical, and legal impacts of computer science. These include security concerns, issues of software ownership and liability, the social impact of database technology, and the consequences of advances in artificial intelligence.

Moreover, there is often no definitive correct answer to a problem, and many valid solutions are compromises between opposing (and perhaps equally valid) views. Finding solutions in these cases often requires the ability to listen, to recognize other points of view, to carry on a rational debate, and to alter one's own opinion as new insights are gained. Thus, each chapter of this text ends with a collection of questions under the heading "Social Issues" that investigate the relationship between computer science and society. These are not necessarily questions to be answered. Instead, they are questions to be considered. In many cases, an answer that may appear obvious at first will cease to satisfy you as you explore alternatives. In short, the purpose of these questions is not to lead you to a "correct" answer, but rather to increase your awareness, including your awareness of the various stakeholders in an issue, your awareness of alternatives, and your awareness of both the short- and long-term consequences of those alternatives.

Philosophers have introduced many approaches to ethics in their search for fundamental theories that lead to principles for guiding decisions and behavior.

Character-based ethics (sometimes called virtue ethics) were promoted by Plato and Aristotle, who argued that "good behavior" is not the result of applying identifiable rules, but instead is a natural consequence of "good character." Whereas other ethical bases, such as consequence-based ethics, duty-based ethics, and contract-based ethics, propose that a person resolve an ethical dilemma by asking, "What are the consequences?", "What are my duties?", or "What contracts do I have?", character-based ethics proposes that dilemmas be resolved by asking, "Who do I want to be?" Thus, good behavior is obtained by building good character, which is typically the result of sound upbringing and the development of virtuous habits.

It is character-based ethics that underlies the approach normally taken when "teaching" ethics to professionals in various fields. Rather than presenting specific ethical theories, the approach is to introduce case studies that expose a variety of ethical questions in the professionals' area of expertise. Then, by discussing the pros and cons in these cases, the professionals become more aware, insightful, and sensitive to the perils lurking in their professional lives and thus grow in character. This is the spirit in which the questions regarding social issues at the end of each chapter are presented.

Social Issues

The following questions are intended as a guide to the ethical/social/legal issues associated with the field of computing. The goal is not merely to answer these questions. You should also consider why you answered as you did and whether your justifications are consistent from one question to the next.

1. The premise that our society is *different* from what it would have been without the computer revolution is generally accepted. Is our society *better* than it would have been without the revolution? Is our society *worse*? Would your answer differ if your position within society were different?
2. Is it acceptable to participate in today's technical society without making an effort to understand the basics of that technology? For instance, do members of a democracy, whose votes often determine how technology will be supported and used, have an obligation to try to understand that technology?

Does your answer depend on which technology is being considered? For example, is your answer the same when considering nuclear technology as when considering computer technology?

3. By using cash in financial transactions, individuals have traditionally had the option to manage their financial affairs without service charges. However, as more of our economy is becoming automated, financial institutions are implementing service charges for access to these automated systems. Is there a point at which these charges unfairly restrict an individual's access to the economy? For example, suppose an employer pays employees only by check, and all financial institutions were to place a service charge on check cashing and depositing. Would the employees be unfairly treated? What if an employer insists on paying only via direct deposit?
4. In the context of interactive television, to what extent should a company be allowed to retrieve information from children (perhaps via an interactive game format)? For example, should a company be allowed to obtain a child's report on his or her parents' buying patterns? What about information about the child?
5. To what extent should a government regulate computer technology and its applications? Consider, for example, the issues mentioned in questions 3 and 4. What justifies governmental regulation?
6. To what extent will our decisions regarding technology in general, and computer technology in particular, affect future generations?
7. As technology advances, our educational system is constantly challenged to reconsider the level of abstraction at which topics are presented. Many questions take the form of whether a skill is still necessary or whether students should be allowed to rely on an abstract tool. Students of trigonometry are no longer taught how to find the values of trigonometric functions using tables. Instead, they use calculators as abstract tools to find these values. Some argue that long division should also give way to abstraction. What other subjects are involved with similar controversies? Do modern word processors eliminate the need to develop spelling skills? Will the use of video technology someday remove the need to read?
8. The concept of public libraries is largely based on the premise that all citizens in a democracy must have access to information. As more information is stored and disseminated via computer technology, does access to this technology become a right of every individual? If so, should public libraries be the channel by which this access is provided?
9. What ethical concerns arise in a society that relies on the use of abstract tools? Are there cases in which it is unethical to use a product or service without understanding how it works? Without knowing how it is produced? Or, without understanding the byproducts of its use?
10. As our society becomes more automated, it becomes easier for governments to monitor their citizens' activities. Is that good or bad?
11. Which technologies that were imagined by George Orwell (Eric Blair) in his novel *1984* have become reality? Are they being used in the manner in which Orwell predicted?
12. If you had a time machine, in which period of history would you like to live? Are there current technologies that you would like to take with you? Could your choice of technologies be taken with you without taking others? To what

extent can one technology be separated from another? Is it consistent to protest against global warming yet accept modern medical treatment?

13. Suppose your job requires that you reside in another culture. Should you continue to practice the ethics of your native culture or adopt the ethics of your host culture? Does your answer depend on whether the issue involves dress code or human rights? Which ethical standards should prevail if you continue to reside in your native culture but conduct business with a foreign culture on the Internet?
14. Has society become too dependent on computer applications for commerce, communications, or social interactions? For example, what would be the consequences of a long-term interruption in Internet and/or cellular telephone service?
15. Most smartphones are able to identify the phone's location by means of GPS. This allows applications to provide location-specific information (such as the local news, local weather, or the presence of businesses in the immediate area) based on the phone's current location. However, such GPS capabilities may also allow other applications to broadcast the phone's location to other parties. Is this good? How could knowledge of the phone's location (thus your location) be abused?

Additional Reading

Goldstine, J. J. *The Computer from Pascal to von Neumann*. Princeton, NJ: Princeton University Press, 1972.

Kizza, J. M. *Ethical and Social Issues in the Information Age*, 3rd ed. London: Springer-Verlag, 2007.

Mollenhoff, C. R. *Atanasoff: Forgotten Father of the Computer*. Ames, IA: Iowa State University Press, 1988.

Neumann, P. G. *Computer Related Risks*. Boston, MA: Addison-Wesley, 1995.

Ni, L. *Smart Phone and Next Generation Mobile Computing*. San Francisco, CA: Morgan Kaufmann, 2006.

Quinn, M. J. *Ethics for the Information Age*, 5th ed. Boston, MA: AddisonWesley, 2012.

Randell, B. *The Origins of Digital Computers*, 3rd ed. New York: SpringerVerlag, 1982.

Spinello, R. A., and H. T. Tavani. *Readings in CyberEthics*, 2nd ed. Sudbury, MA: Jones and Bartlett, 2004.

Swade, D. *The Difference Engine*. New York: Viking, 2000.

Tavani, H. T. *Ethics and Technology: Ethical Issues in an Age of Information and Communication Technology*, 4th ed. New York: Wiley, 2012.

Woolley, B. *The Bride of Science: Romance, Reason, and Byron's Daughter*. New York: McGraw-Hill, 1999.

Data Storage

In this chapter, we consider topics associated with data representation and the storage of data within a computer. The types of data we will consider include text, numeric values, images, audio, and video. Much of the information in this chapter is also relevant to fields other than traditional computing, such as digital photography, audio/video recording and reproduction, and long-distance communication.

1.1 Bits and Their Storage

Boolean Operations
Gates and Flip-Flops
Hexadecimal Notation

1.2 Main Memory

Memory Organization
Measuring Memory Capacity

1.3 Mass Storage

Magnetic Systems
Optical Systems
Flash Drives

1.4 Representing Information as Bit Patterns

Representing Text
Representing Numeric Values

Representing Images
Representing Sound

*1.5 The Binary System

Binary Notation
Binary Addition
Fractions in Binary

*1.6 Storing Integers

Two's Complement Notation
Excess Notation

*1.7 Storing Fractions

Floating-Point Notation
Truncation Errors

*1.8 Data and Programming

Getting Started With Python
Hello Python

Variables
Operators and Expressions
Currency Conversion
Debugging

*1.9 Data Compression

Generic Data Compression Techniques
Compressing Images
Compressing Audio and Video

*1.10 Communication Errors

Parity Bits
Error-Correcting Codes

**Asterisks indicate suggestions for optional sections.*

We begin our study of computer science by considering how information is encoded and stored inside computers. Our first step is to discuss the basics of a computer's data storage devices and then to consider how information is encoded for storage in these systems. We will explore the ramifications of today's data storage systems and how such techniques as data compression and error handling are used to overcome their shortfalls.

1.1 Bits and Their Storage

Inside today's computers information is encoded as patterns of 0s and 1s. These digits are called **bits** (short for *binary digits*). Although you may be inclined to associate bits with numeric values, they are really only symbols whose meaning depends on the application at hand. Sometimes patterns of bits are used to represent numeric values; sometimes they represent characters in an alphabet and punctuation marks; sometimes they represent images; and sometimes they represent sounds.

Boolean Operations

To understand how individual bits are stored and manipulated inside a computer, it is convenient to imagine that the bit 0 represents the value *false* and the bit 1 represents the value *true*. Operations that manipulate true/false values are called **Boolean operations**, in honor of the mathematician George Boole (1815–1864), who was a pioneer in the field of mathematics called logic. Three of the basic Boolean operations are AND, OR, and XOR (exclusive or) as summarized in Figure 1.1. (We capitalize these Boolean operation names to distinguish them from their English word counterparts.) These operations are similar to the arithmetic operations TIMES and PLUS because they combine a pair of values (the operation's input) to produce a third value (the output). In contrast to arithmetic operations, however, Boolean operations combine true/false values rather than numeric values.

The Boolean operation AND is designed to reflect the truth or falseness of a statement formed by combining two smaller, or simpler, statements with the conjunction *and*. Such statements have the generic form

$P \text{ AND } Q$

where P represents one statement, and Q represents another—for example,

Kermit is a frog AND Miss Piggy is an actress.

The inputs to the AND operation represent the truth or falseness of the compound statement's components; the output represents the truth or falseness of the compound statement itself. Since a statement of the form $P \text{ AND } Q$ is true only when both of its components are true, we conclude that 1 AND 1 should be 1, whereas all other cases should produce an output of 0, in agreement with Figure 1.1.

In a similar manner, the OR operation is based on compound statements of the form

$P \text{ OR } Q$

where, again, P represents one statement and Q represents another. Such statements are true when at least one of their components is true, which agrees with the OR operation depicted in Figure 1.1.

Figure 1.1 The possible input and output values of Boolean operations AND, OR, and XOR (exclusive or)

| The AND operation | | | |
|--|--|--|--|
| $\begin{array}{r} 0 \\ \text{AND} \\ \hline 0 \end{array}$ | $\begin{array}{r} 0 \\ \text{AND} \\ \hline 0 \end{array}$ | $\begin{array}{r} 1 \\ \text{AND} \\ \hline 0 \end{array}$ | $\begin{array}{r} 1 \\ \text{AND} \\ \hline 1 \end{array}$ |
| | | | |
| The OR operation | | | |
| $\begin{array}{r} 0 \\ \text{OR} \\ \hline 0 \end{array}$ | $\begin{array}{r} 0 \\ \text{OR} \\ \hline 1 \end{array}$ | $\begin{array}{r} 1 \\ \text{OR} \\ \hline 0 \end{array}$ | $\begin{array}{r} 1 \\ \text{OR} \\ \hline 1 \end{array}$ |
| | | | |
| The XOR operation | | | |
| $\begin{array}{r} 0 \\ \text{XOR} \\ \hline 0 \end{array}$ | $\begin{array}{r} 0 \\ \text{XOR} \\ \hline 1 \end{array}$ | $\begin{array}{r} 1 \\ \text{XOR} \\ \hline 0 \end{array}$ | $\begin{array}{r} 1 \\ \text{XOR} \\ \hline 1 \end{array}$ |
| | | | |

There is not a single conjunction in the English language that captures the meaning of the XOR operation. XOR produces an output of 1 (true) when one of its inputs is 1 (true) and the other is 0 (false). For example, a statement of the form $P \text{ XOR } Q$ means “either P or Q but not both.” (In short, the XOR operation produces an output of 1 when its inputs are different.)

The operation NOT is another Boolean operation. It differs from AND, OR, and XOR because it has only one input. Its output is the opposite of that input; if the input of the operation NOT is true, then the output is false, and vice versa. Thus, if the input of the NOT operation is the truth or falseness of the statement

Fozzie is a bear.

then the output would represent the truth or falseness of the statement

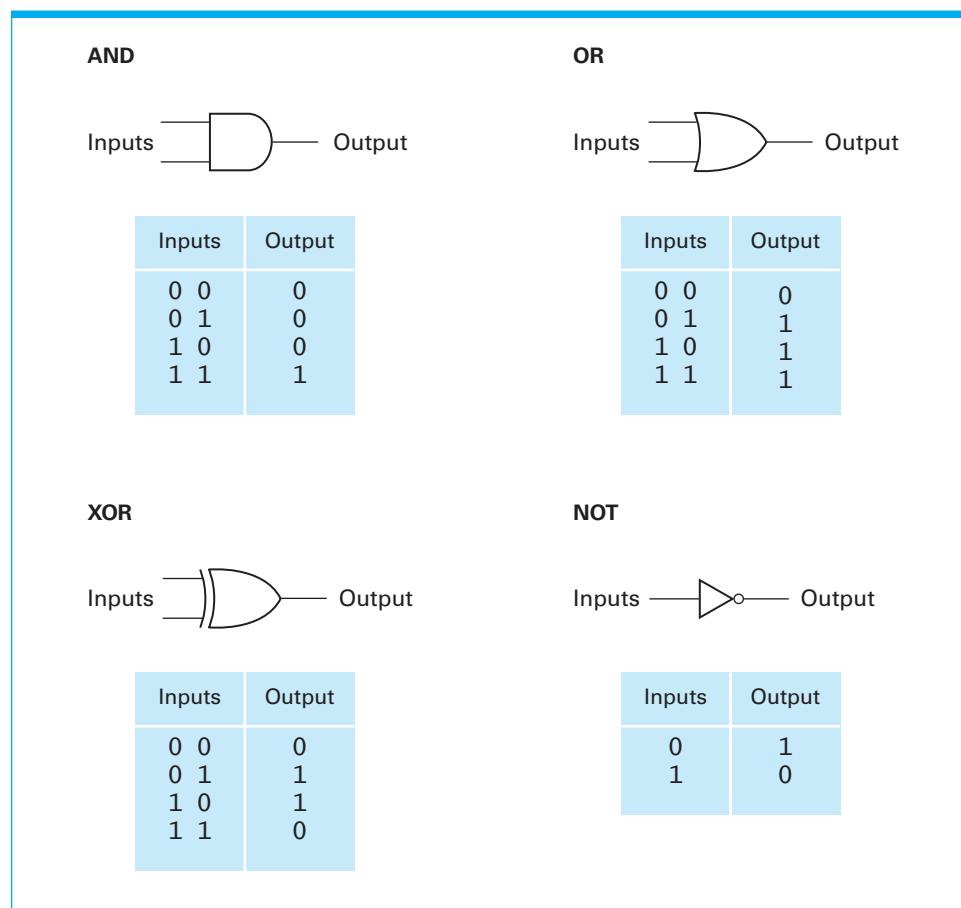
Fozzie is not a bear.

Gates and Flip-Flops

A device that produces the output of a Boolean operation when given the operation’s input values is called a **gate**. Gates can be constructed from a variety of technologies such as gears, relays, and optic devices. Inside today’s computers, gates are usually implemented as small electronic circuits in which the digits 0 and 1 are represented as voltage levels. We need not concern ourselves with such details, however. For our purposes, it suffices to represent gates in their symbolic form, as shown in Figure 1.2. Note that the AND, OR, XOR, and NOT gates are represented by distinctively shaped symbols, with the input values entering on one side, and the output exiting on the other.

Gates provide the building blocks from which computers are constructed. One important step in this direction is depicted in the circuit in Figure 1.3. This is a particular example from a collection of circuits known as a **flip-flop**. A flip-flop

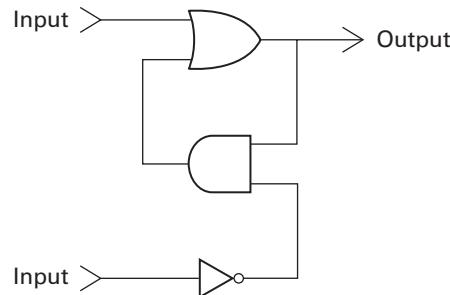
Figure 1.2 A pictorial representation of AND, OR, XOR, and NOT gates as well as their input and output values



is a fundamental unit of computer memory. It is a circuit that produces an output value of 0 or 1, which remains constant until a pulse (a temporary change to a 1 that returns to 0) from another circuit causes it to shift to the other value. In other words, the output can be set to “remember” a zero or a one under control of external stimuli. As long as both inputs in the circuit in Figure 1.3 remain 0, the output (whether 0 or 1) will not change. However, temporarily placing a 1 on the upper input will force the output to be 1, whereas temporarily placing a 1 on the lower input will force the output to be 0.

Let us consider this claim in more detail. Without knowing the current output of the circuit in Figure 1.3, suppose that the upper input is changed to 1 while the lower input remains 0 (Figure 1.4a). This will cause the output of the OR gate to be 1, regardless of the other input to this gate. In turn, both inputs to the AND gate will now be 1, since the other input to this gate is already 1 (the output produced by the NOT gate whenever the lower input of the flip-flop is at 0). The output of the AND gate will then become 1, which means that the second input to the OR gate will now be 1 (Figure 1.4b). This guarantees that the output of the OR gate will remain 1, even when the upper input to the flip-flop is changed back to 0 (Figure 1.4c). In summary, the flip-flop’s output has become 1, and this output value will remain after the upper input returns to 0.

Figure 1.3 A simple flip-flop circuit

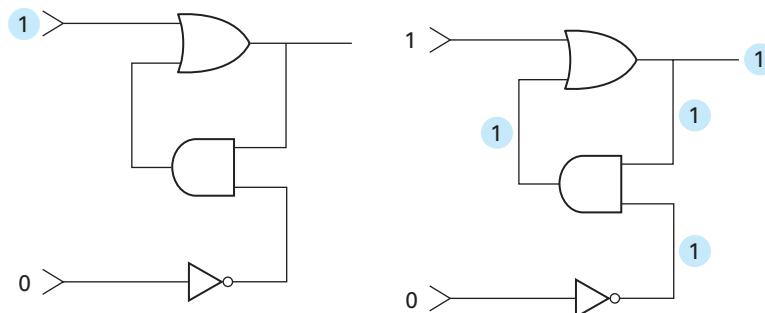


In a similar manner, temporarily placing the value 1 on the lower input will force the flip-flop's output to be 0, and this output will persist after the input value returns to 0.

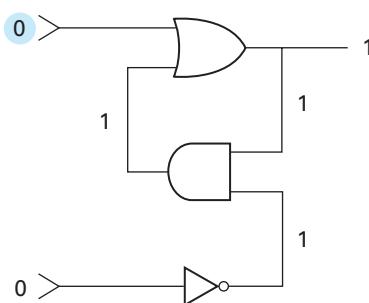
Our purpose in introducing the flip-flop circuit in Figures 1.3 and 1.4 is threefold. First, it demonstrates how devices can be constructed from gates, a process known as digital circuit design, which is an important topic in computer

Figure 1.4 Setting the output of a flip-flop to 1

- a. First, a 1 is placed on the upper input.
- b. This causes the output of the OR gate to be 1 and, in turn, the output of the AND gate to be 1.



- c. Finally, the 1 from the AND gate keeps the OR gate from changing after the upper input returns to 0.



engineering. Indeed, the flip-flop is only one of many circuits that are basic tools in computer engineering.

Second, the concept of a flip-flop provides an example of abstraction and the use of abstract tools. Actually, there are other ways to build a flip-flop. One alternative is shown in Figure 1.5. If you experiment with this circuit, you will find that, although it has a different internal structure, its external properties are the same as those of Figure 1.3. A computer engineer does not need to know which circuit is actually used within a flip-flop. Instead, only an understanding of the flip-flop's external properties is needed to use it as an abstract tool. A flip-flop, along with other well-defined circuits, forms a set of building blocks from which an engineer can construct more complex circuitry. In turn, the design of computer circuitry takes on a hierarchical structure, each level of which uses the lower level components as abstract tools.

The third purpose for introducing the flip-flop is that it is one means of storing a bit within a modern computer. More precisely, a flip-flop can be set to have the output value of either 0 or 1. Other circuits can adjust this value by sending pulses to the flip-flop's inputs, and still other circuits can respond to the stored value by using the flip-flop's output as their inputs. Thus, many flip-flops, constructed as very small electrical circuits, can be used inside a computer as a means of recording information that is encoded as patterns of 0s and 1s. Indeed, technology known as **very large-scale integration (VLSI)**, which allows millions of electrical components to be constructed on a wafer (called a **chip**), is used to create miniature devices containing millions of flip-flops along with their controlling circuitry. Consequently, these chips are used as abstract tools in the construction of computer systems. In fact, in some cases VLSI is used to create an entire computer system on a single chip.

Hexadecimal Notation

When considering the internal activities of a computer, we must deal with patterns of bits, which we will refer to as a string of bits, some of which can be quite long. A long string of bits is often called a **stream**. Unfortunately, streams are difficult for the human mind to comprehend. Merely transcribing the pattern 101101010011 is tedious and error prone. To simplify the representation of such bit patterns, therefore, we usually use a shorthand notation called **hexadecimal notation**, which takes advantage of the fact that bit patterns within a machine

Figure 1.5 Another way of constructing a flip-flop

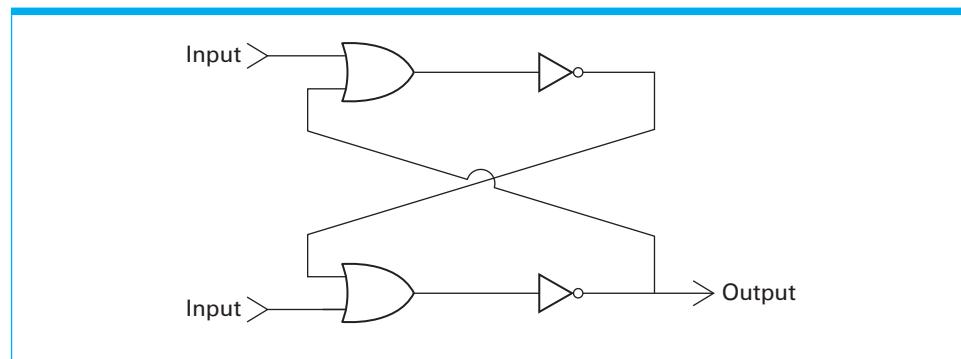


Figure 1.6 The hexadecimal encoding system

| Bit pattern | Hexadecimal representation |
|-------------|----------------------------|
| 0000 | 0 |
| 0001 | 1 |
| 0010 | 2 |
| 0011 | 3 |
| 0100 | 4 |
| 0101 | 5 |
| 0110 | 6 |
| 0111 | 7 |
| 1000 | 8 |
| 1001 | 9 |
| 1010 | A |
| 1011 | B |
| 1100 | C |
| 1101 | D |
| 1110 | E |
| 1111 | F |

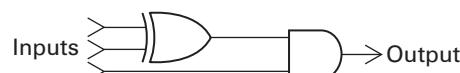
tend to have lengths in multiples of four. In particular, hexadecimal notation uses a single symbol to represent a pattern of four bits. For example, a string of twelve bits can be represented by three hexadecimal symbols.

Figure 1.6 presents the hexadecimal encoding system. The left column displays all possible bit patterns of length four; the right column shows the symbol used in hexadecimal notation to represent the bit pattern to its left. Using this system, the bit pattern 10110101 is represented as B5. This is obtained by dividing the bit pattern into substrings of length four and then representing each substring by its hexadecimal equivalent—1011 is represented by B, and 0101 is represented by 5. In this manner, the 16-bit pattern 1010010011001000 can be reduced to the more palatable form A4C8.

We will use hexadecimal notation extensively in the next chapter. There you will come to appreciate its efficiency.

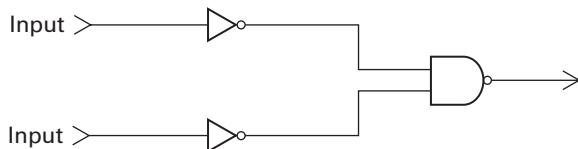
Questions & Exercises

- What input bit patterns will cause the following circuit to produce an output of 1?

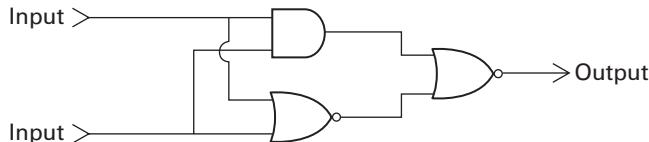


- In the text, we claimed that placing a 1 on the lower input of the flip-flop in Figure 1.3 (while holding the upper input at 0) will force the flip-flop's output to be 0. Describe the sequence of events that occurs within the flip-flop in this case.

3. Assuming that both inputs to the flip-flop in Figure 1.5 begin as 0, describe the sequence of events that occurs when the upper input is temporarily set to 1.
4. a. If the output of an AND gate is passed through a NOT gate, the combination computes the Boolean operation called NAND, which has an output of 0 only when both its inputs are 1. The symbol for a NAND gate is the same as an AND gate except that it has a circle at its output. The following is a circuit containing a NAND gate. What Boolean operation does the circuit compute?



- b. If the output of an OR gate is passed through a NOT gate, the combination computes the Boolean operation called NOR that has an output of 1 only when both its inputs are 0. The symbol for a NOR gate is the same as an OR gate except that it has a circle at its output. The following is a circuit containing an AND gate and two NOR gates. What Boolean operation does the circuit compute?



5. Use hexadecimal notation to represent the following bit patterns:
- a. 0110101011110010
 - b. 1110100001010100010111
 - c. 01001000
6. What bit patterns are represented by the following hexadecimal patterns?
- a. 5FD97
 - b. 610A
 - c. ABCD
 - d. 0100

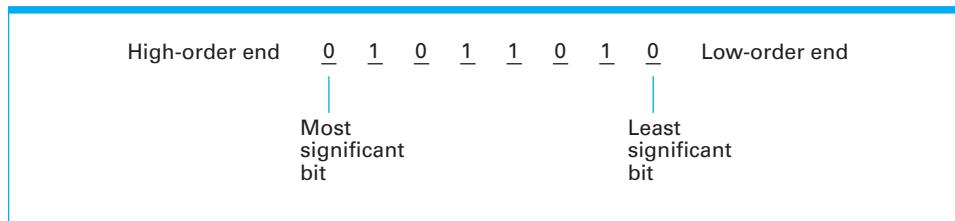
1.2 Main Memory

For the purpose of storing data, a computer contains a large collection of circuits (such as flip-flops), each capable of storing a single bit. This bit reservoir is known as the machine's **main memory**.

Memory Organization

A computer's main memory is organized in manageable units called **cells**, with a typical cell size being eight bits. (A string of eight bits is called a **byte**. Thus, a typical memory cell has a capacity of one byte.) Small computers embedded in such household devices as microwave ovens may have main memories consisting

Figure 1.7 The organization of a byte-size memory cell



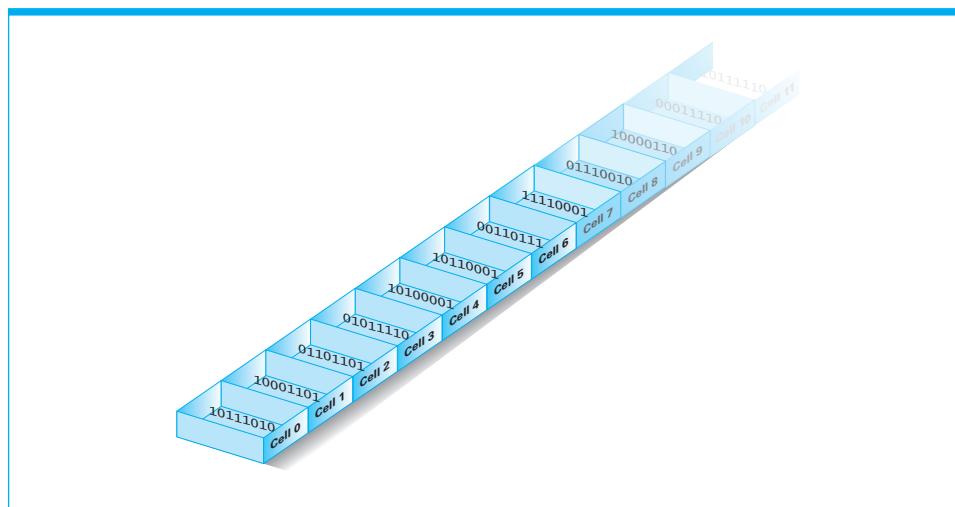
of only a few hundred cells, whereas large computers may have billions of cells in their main memories.

Although there is no left or right within a computer, we normally envision the bits within a memory cell as being arranged in a row. The left end of this row is called the **high-order end**, and the right end is called the **low-order end**. The leftmost bit is called either the high-order bit or the **most significant bit** in reference to the fact that if the contents of the cell were interpreted as representing a numeric value, this bit would be the most significant digit in the number. Similarly, the rightmost bit is referred to as the low-order bit or the **least significant bit**. Thus we may represent the contents of a byte-size memory cell as shown in Figure 1.7.

To identify individual cells in a computer's main memory, each cell is assigned a unique "name," called its **address**. The system is analogous to the technique of identifying houses in a city by addresses. In the case of memory cells, however, the addresses used are entirely numeric. To be more precise, we envision all the cells being placed in a single row and numbered in this order starting with the value zero. Such an addressing system not only gives us a way of uniquely identifying each cell but also associates an order to the cells (Figure 1.8), giving us phrases such as "the next cell" or "the previous cell."

An important consequence of assigning an order to both the cells in main memory and the bits within each cell is that the entire collection of bits within a computer's main memory is essentially ordered in one long row. Pieces of this long row can therefore be used to store bit patterns that may be longer than the

Figure 1.8 Memory cells arranged by address



length of a single cell. In particular, we can still store a string of 16 bits merely by using two consecutive memory cells.

To complete the main memory of a computer, the circuitry that actually holds the bits is combined with the circuitry required to allow other circuits to store and retrieve data from the memory cells. In this way, other circuits can get data from the memory by electronically asking for the contents of a certain address (called a read operation), or they can record information in the memory by requesting that a certain bit pattern be placed in the cell at a particular address (called a write operation).

Because a computer's main memory is organized as individual, addressable cells, the cells can be accessed independently as required. To reflect the ability to access cells in any order, a computer's main memory is often called **random access memory (RAM)**. This random access feature of main memory is in stark contrast to the mass storage systems that we will discuss in the next section, in which long strings of bits are manipulated as amalgamated blocks.

Although we have introduced flip-flops as a means of storing bits, the RAM in most modern computers is constructed using analogous, but more complex technologies that provide greater miniaturization and faster response time. Many of these technologies store bits as tiny electric charges that dissipate quickly. Thus these devices require additional circuitry, known as a refresh circuit, that repeatedly replenishes the charges many times a second. In recognition of this volatility, computer memory constructed from such technology is often called **dynamic memory**, leading to the term **DRAM** (pronounced "DEE-ram") meaning Dynamic RAM. Or, at times the term **SDRAM** (pronounced "ES-DEE-ram"), meaning Synchronous DRAM, is used in reference to DRAM that applies additional techniques to decrease the time needed to retrieve the contents from its memory cells.

Measuring Memory Capacity

As we will learn in the next chapter, it is convenient to design main memory systems in which the total number of cells is a power of two. In turn, the size of the memories in early computers were often measured in 1024 (which is 2^{10}) cell units. Since 1024 is close to the value 1000, the computing community adopted the prefix *kilo* in reference to this unit. That is, the term *kilobyte* (abbreviated KB) was used to refer to 1024 bytes. Thus, a machine with 4096 memory cells was said to have a 4KB memory ($4096 = 4 \times 1024$). As memories became larger, this terminology grew to include MB (megabyte), GB (gigabyte), and TB (terabyte). Unfortunately, this application of prefixes *kilo-*, *mega-*, and so on, represents a misuse of terminology because these are already used in other fields in reference to units that are powers of a thousand. For example, when measuring distance, *kilometer* refers to 1000 meters, and when measuring radio frequencies, *megahertz* refers to 1,000,000 hertz. In the late 1990s, international standards organizations developed specialized terminology for powers of two: *kibi-*, *mebi-*, *gibi-*, and *tebi*-bytes denote powers of 1024, rather than powers of a thousand. However, while this distinction is the law of the land in many parts of the world, both the general public and many computer scientists have been reluctant to abandon the more familiar, yet ambiguous "megabyte." Thus, a word of caution is in order when using this terminology. As a general rule, terms such as *kilo-*, *mega-*, etc. refer to powers of two when used in the context of computer measurements, but they refer to powers of a thousand when used in other contexts.

Questions & Exercises

1. If the memory cell whose address is 5 contains the value 8, what is the difference between writing the value 5 into cell number 6 and moving the contents of cell number 5 into cell number 6?
2. Suppose you want to interchange the values stored in memory cells 2 and 3. What is wrong with the following sequence of steps:
Step 1. Move the contents of cell number 2 to cell number 3.
Step 2. Move the contents of cell number 3 to cell number 2.
Design a sequence of steps that correctly interchanges the contents of these cells. If needed, you may use additional cells.
3. How many bits would be in the memory of a computer with 4KB memory?

1.3 Mass Storage

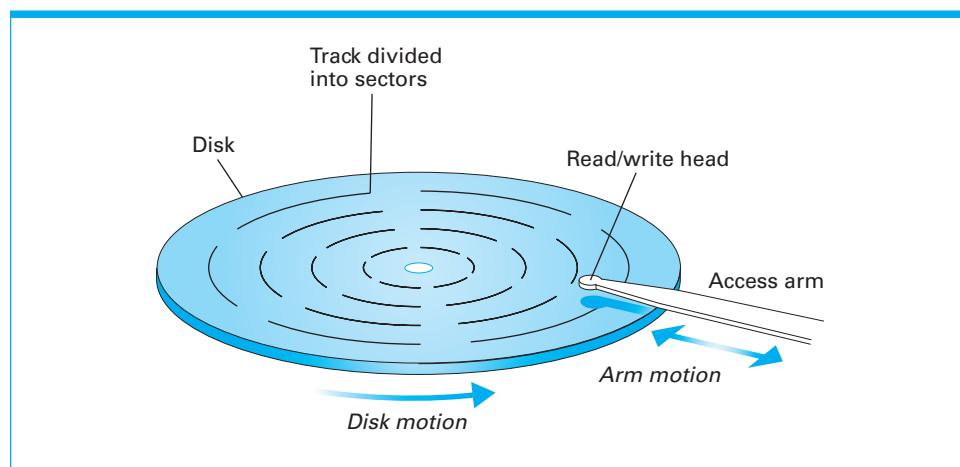
Due to the volatility and limited size of a computer's main memory, most computers have additional memory devices called **mass storage** (or secondary storage) systems, including magnetic disks, CDs, DVDs, magnetic tapes, flash drives, and solid-state disks (all of which we will discuss shortly). The advantages of mass storage systems over main memory include less volatility, large storage capacities, low cost, and in many cases, the ability to remove the storage medium from the machine for archival purposes.

A major disadvantage of magnetic and optical mass storage systems is that they typically require mechanical motion and therefore require significantly more time to store and retrieve data than a machine's main memory, where all activities are performed electronically. Moreover, storage systems with moving parts are more prone to mechanical failures than solid state systems.

Magnetic Systems

For years, magnetic technology has dominated the mass storage arena. The most common example in use today is the **magnetic disk** or **hard disk drive (HDD)**, in which a thin spinning disk with magnetic coating is used to hold data (Figure 1.9). Read/write heads are placed above and/or below the disk so that as the disk spins, each head traverses a circle, called a **track**. By repositioning the read/write heads, different concentric tracks can be accessed. In many cases, a disk storage system consists of several disks mounted on a common spindle, one on top of the other, with enough space for the read/write heads to slip between the platters. In such cases, the read/write heads move in unison. Each time the read/write heads are repositioned, a new set of tracks—which is called a **cylinder**—becomes accessible.

Since a track can contain more information than we would normally want to manipulate at any one time, each track is divided into small arcs called **sectors** on which information is recorded as a continuous string of bits. All sectors on a disk contain the same number of bits (typical capacities are in the range of 512 bytes to a few KB), and in the simplest disk storage systems each track contains the same

Figure 1.9 A disk storage system

number of sectors. Thus, the bits within a sector on a track near the outer edge of the disk are less compactly stored than those on the tracks near the center, since the outer tracks are longer than the inner ones. In contrast, in high-capacity disk storage systems, the tracks near the outer edge are capable of containing significantly more sectors than those near the center, and this capability is often used by applying a technique called **zoned-bit recording**. Using zoned-bit recording, several adjacent tracks are collectively known as zones, with a typical disk containing approximately 10 zones. All tracks within a zone have the same number of sectors, but each zone has more sectors per track than the zone inside of it. In this manner, efficient use of the entire disk surface is achieved. Regardless of the details, a disk storage system consists of many individual sectors, each of which can be accessed as an independent string of bits.

The capacity of a disk storage system depends on the number of platters used and the density in which the tracks and sectors are placed. Lower-capacity systems may consist of a single platter. High-capacity disk systems, capable of holding many gigabytes, or even terabytes, consist of perhaps three to six platters mounted on a common spindle. Furthermore, data may be stored on both the upper and lower surfaces of each platter.

Several measurements are used to evaluate a disk system's performance: (1) **seek time** (the time required to move the read/write heads from one track to another); (2) **rotation delay or latency time** (half the time required for the disk to make a complete rotation, which is the average amount of time required for the desired data to rotate around to the read/write head once the head has been positioned over the desired track); (3) **access time** (the sum of seek time and rotation delay); and (4) **transfer rate** (the rate at which data can be transferred to or from the disk). (Note that in the case of zone-bit recording, the amount of data passing a read/write head in a single disk rotation is greater for tracks in an outer zone than for an inner zone, and therefore the data transfer rate varies depending on the portion of the disk being used.)

A factor limiting the access time and transfer rate is the speed at which a disk system rotates. To facilitate fast rotation speeds, the read/write heads in these systems do not touch the disk but instead "float" just off the surface. The spacing is so close that even a single particle of dust could become jammed

between the head and disk surface, destroying both (a phenomenon known as a head crash). Thus, disk systems are typically housed in cases that are sealed at the factory. With this construction, disk systems are able to rotate at speeds of several hundred times per second, achieving transfer rates that are measured in MB per second.

Since disk systems require physical motion for their operation, these systems suffer when compared to speeds within electronic circuitry. Delay times within an electronic circuit are measured in units of nanoseconds (billions of a second) or less, whereas seek times, latency times, and access times of disk systems are measured in milliseconds (thousandths of a second). Thus the time required to retrieve information from a disk system can seem like an eternity to an electronic circuit awaiting a result.

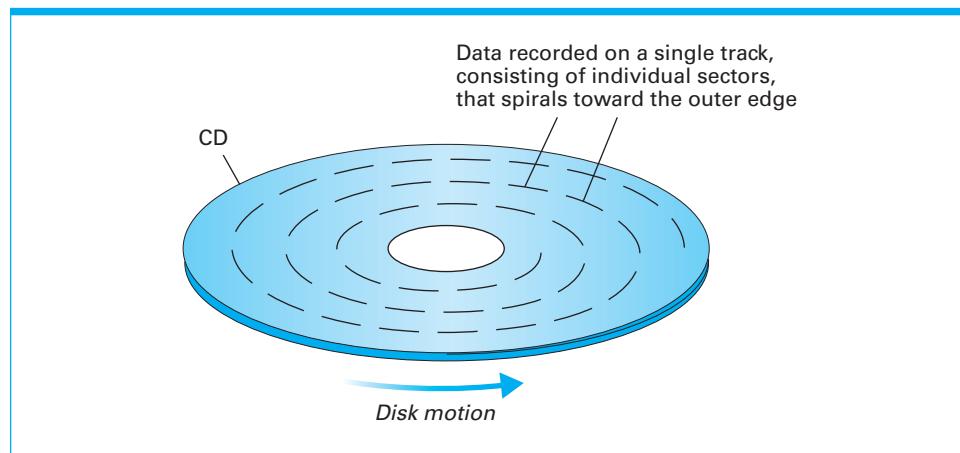
Magnetic storage technologies that are now less widely used include **magnetic tape**, in which information is recorded on the magnetic coating of a thin plastic tape wound on reels, and **floppy disk drives**, in which single platters with a magnetic coating are encased in a portable cartridge designed to be readily removed from the drive. Magnetic tape drives have extremely long seek times, just as their cousins, audio cassettes, suffer from long rewind and fast-forward times. Low cost and high data capacities still make magnetic tape suitable for applications where data is primarily read or written linearly, such as archival data backups. The removable nature of floppy disk platters came at the cost of much lower data densities and access speeds than hard disk platters, but their portability was extremely valuable in earlier decades, prior to the arrival of flash drives with larger capacity and higher durability.

Optical Systems

Another class of mass storage systems applies optical technology. An example is the **compact disk (CD)**. These disks are 12 centimeters (approximately 5 inches) in diameter and consist of reflective material covered with a clear protective coating. Information is recorded on them by creating variations in their reflective surfaces. This information can then be retrieved by means of a laser that detects irregularities on the reflective surface of the CD as it spins.

CD technology was originally applied to audio recordings using a recording format known as **CD-DA (compact disk-digital audio)**, and the CDs used today for computer data storage use essentially the same format. In particular, information on these CDs is stored on a single track that spirals around the CD like a groove in an old-fashioned phonograph record, however, unlike old-fashioned phonograph records, the track on a CD spirals from the inside out (Figure 1.10). This track is divided into units called sectors, each with its own identifying markings and a capacity of 2KB of data, which equates to 1/75 of a second of music in the case of audio recordings.

Note that the distance around the spiraled track is greater toward the outer edge of the disk than at the inner portion. To maximize the capacity of a CD, information is stored at a uniform linear density over the entire spiraled track, which means that more information is stored in a loop around the outer portion of the spiral than in a loop around the inner portion. In turn, more sectors will be read in a single revolution of the disk when the laser is scanning the outer portion of the spiraled track than when the laser is scanning the inner portion of the track. Thus, to obtain a uniform rate of data transfer, CD-DA players are designed

Figure 1.10 CD storage format

to vary the rotation speed depending on the location of the laser. However, most CD systems used for computer data storage spin at a faster, constant speed and thus must accommodate variations in data transfer rates.

As a consequence of such design decisions, CD storage systems perform best when dealing with long, continuous strings of data, as when reproducing music. In contrast, when an application requires access to items of data in a random manner, the approach used in magnetic disk storage (individual, concentric tracks divided into individually accessible sectors) outperforms the spiral approach used in CDs.

Traditional CDs have capacities in the range of 600 to 700MB. However, **DVDs (Digital Versatile Disks)**, which are constructed from multiple, semi-transparent layers that serve as distinct surfaces when viewed by a precisely focused laser, provide storage capacities of several GB. Such disks are capable of storing lengthy multimedia presentations, including entire motion pictures. Finally, Blu-ray technology, which uses a laser in the blue-violet spectrum of light (instead of red), is able to focus its laser beam with very fine precision. As a result, **BDs (Blu-ray Disks)** provides over five times the capacity of a DVD. This seemingly vast amount of storage is needed to meet the demands of high definition video.

Flash Drives

A common property of mass storage systems based on magnetic or optic technology is that physical motion, such as spinning disks, moving read/write heads, and aiming laser beams, is required to store and retrieve data. This means that data storage and retrieval is slow compared to the speed of electronic circuitry. **Flash memory** technology has the potential of alleviating this drawback. In a flash memory system, bits are stored by sending electronic signals directly to the storage medium where they cause electrons to be trapped in tiny chambers of silicon dioxide, thus altering the characteristics of small electronic circuits. Since these chambers are able to hold their captive electrons for many years without external power, this technology is excellent for portable, nonvolatile data storage.

Although data stored in flash memory systems can be accessed in small byte-size units as in RAM applications, current technology dictates that stored data be erased in large blocks. Moreover, repeated erasing slowly damages the silicon dioxide chambers, meaning that current flash memory technology is not suitable for general main memory applications where its contents might be altered many times a second. However, in those applications in which alterations can be controlled to a reasonable level, such as in digital cameras and smartphones, flash memory has become the mass storage technology of choice. Indeed, since flash memory is not sensitive to physical shock (in contrast to magnetic and optic systems), it is now replacing other mass storage technologies in portable applications such as laptop computers.

Flash memory devices called **flash drives**, with capacities of hundreds of GBs, are available for general mass storage applications. These units are packaged in ever smaller plastic cases with a removable cap on one end to protect the unit's electrical connector when the drive is offline. The high capacity of these portable units as well as the fact that they are easily connected to and disconnected from a computer make them ideal for portable data storage. However, the vulnerability of their tiny storage chambers dictates that they are not as reliable as optical disks for truly long-term applications.

Larger flash memory devices called **SSDs (solid-state disks)** are explicitly designed to take the place of magnetic hard disks. SSDs compare favorably to hard disks in their resilience to vibrations and physical shock, their quiet operation (due to no moving parts), and their lower access times. SSDs remain more expensive than hard disks of comparable size and thus are still considered a high-end option when buying a computer. SSD sectors suffer from the more limited lifetime of all flash memory technologies, but the use of **wear-leveling** techniques can reduce the impact of this by relocating frequently altered data blocks to fresh locations on the drive.

Another application of flash technology is found in **SD (Secure Digital) memory cards** (or just SD Card). These provide up to two GBs of storage and are packaged in a plastic rigged wafer about the size a postage stamp (SD cards are also available in smaller mini and micro sizes). **SDHC (High Capacity)** memory cards can provide up to 32 GBs and the next generation **SDXC (Extended Capacity) memory cards** may exceed a TB. Given their compact physical size, these cards conveniently slip into slots of small electronic devices. Thus, they are ideal for digital cameras, smartphones, music players, car navigation systems, and a host of other electronic appliances.

Questions & Exercises

1. What is gained by increasing the rotation speed of a disk or CD?
2. When recording data on a multiple-disk storage system, should we fill a complete disk surface before starting on another surface, or should we first fill an entire cylinder before starting on another cylinder?
3. Why should the data in a reservation system that is constantly being updated be stored on a magnetic disk instead of a CD or DVD?

4. What factors allow CD, DVD, and Blu-ray disks all to be read by the same drive?
5. What advantage do flash drives have over the other mass storage systems introduced in this section?
6. What advantages continue to make magnetic hard disk drives competitive?

1.4 Representing Information as Bit Patterns

Having considered techniques for storing bits, we now consider how information can be encoded as bit patterns. Our study focuses on popular methods for encoding text, numerical data, images, and sound. Each of these systems has repercussions that are often visible to a typical computer user. Our goal is to understand enough about these techniques so that we can recognize their consequences for what they are.

Representing Text

Information in the form of text is normally represented by means of a code in which each of the different symbols in the text (such as the letters of the alphabet and punctuation marks) is assigned a unique bit pattern. The text is then represented as a long string of bits in which the successive patterns represent the successive symbols in the original text.

In the 1940s and 1950s, many such codes were designed and used in connection with different pieces of equipment, producing a corresponding proliferation of communication problems. To alleviate this situation, the **American National Standards Institute** (ANSI, pronounced “AN-see”) adopted the **American Standard Code for Information Interchange** (ASCII, pronounced “AS-kee”). This code uses bit patterns of length seven to represent the uppercase and lowercase letters of the English alphabet, punctuation symbols, the digits 0 through 9, and certain control information such as line feeds, carriage returns, and tabs. ASCII is extended to an eight-bit-per-symbol format by adding a 0 at the most significant end of each of the seven-bit patterns. This technique not only produces a code in which each pattern fits conveniently into a typical byte-size memory cell but also provides 128 additional bit patterns (those obtained by assigning the extra bit the value 1) that can be used to represent symbols beyond the English alphabet and associated punctuation.

A portion of ASCII in its eight-bit-per-symbol format is shown in Appendix A. By referring to this appendix, we can decode the bit pattern

01001000 01100101 01101100 01101100 01101111 00101110

as the message “Hello.” as demonstrated in Figure 1.11.

The **International Organization for Standardization** (also known as ISO, in reference to the Greek word *isos*, meaning equal) has developed a number of extensions to ASCII, each of which was designed to accommodate a major language group. For example, one standard provides the symbols needed to express the text of most Western European languages. Included in its 128 additional patterns are symbols for the British pound and the German vowels ä, ö, and ü.

Figure 1.11 The message “Hello.” in ASCII or UTF-8 encoding

| | | | | | |
|----------|----------|----------|----------|----------|----------|
| 01001000 | 01100101 | 01101100 | 01101100 | 01101111 | 00101110 |
| H | e | I | I | o | . |

The ISO-extended ASCII standards made tremendous headway toward supporting all of the world’s multilingual communication; however, two major obstacles surfaced. First, the number of extra bit patterns available in extended ASCII is simply insufficient to accommodate the alphabet of many Asian and some Eastern European languages. Second, because a given document was constrained to using symbols in just the one selected standard, documents containing text of languages from disparate language groups could not be supported. Both proved to be a significant detriment to international use. To address this deficiency, **Unicode** was developed through the cooperation of several of the leading manufacturers of hardware and software and has rapidly gained the support of the computing community. This code uses a unique pattern of up to 21 bits to represent each symbol. When the Unicode character set is combined with the **Unicode Transformation Format 8-bit (UTF-8)** encoding standard, the original ASCII characters can still be represented with 8 bits, while the thousands of additional characters from such languages as Chinese, Japanese, and Hebrew can be represented by 16 bits. Beyond the characters required for all of the world’s commonly used languages, UTF-8 uses 24- or 32-bit patterns to represent more obscure Unicode symbols, leaving ample room for future expansion.

A file consisting of a long sequence of symbols encoded using ASCII or Unicode is often called a **text file**. It is important to distinguish between simple text files that are manipulated by utility programs called **text editors** (or often simply editors) and the more elaborate files produced by **word processors** such as Microsoft’s Word. Both consist of textual material. However, a text file contains only a character-by-character encoding of the text, whereas a file produced by a word processor contains numerous proprietary codes representing changes in fonts, alignment information, and other parameters.

Representing Numeric Values

Storing information in terms of encoded characters is inefficient when the information being recorded is purely numeric. To see why, consider the problem of storing the value 25. If we insist on storing it as encoded symbols in ASCII using one byte per symbol, we need a total of 16 bits. Moreover, the largest number we could store using 16 bits is 99. However, as we will shortly see, by using **binary notation** we can store any integer in the range from 0 to 65535 in these 16 bits. Thus, binary notation (or variations of it) is used extensively for encoded numeric data for computer storage.

Binary notation is a way of representing numeric values using only the digits 0 and 1 rather than the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 as in the traditional decimal, or base 10, system. We will study the binary system more thoroughly in Section 1.5. For now, all we need is an elementary understanding of the system. For this purpose consider an old-fashioned car odometer whose display wheels

The American National Standards Institute

The American National Standards Institute (ANSI) was founded in 1918 by a small consortium of engineering societies and government agencies as a nonprofit federation to coordinate the development of voluntary standards in the private sector. Today, ANSI membership includes more than 1300 businesses, professional organizations, trade associations, and government agencies. ANSI is headquartered in New York and represents the United States as a member body in the ISO. The website for the American National Standards Institute is at <http://www.ansi.org>.

Similar organizations in other countries include Standards Australia (Australia), Standards Council of Canada (Canada), China State Bureau of Quality and Technical Supervision (China), Deutsches Institut für Normung (Germany), Japanese Industrial Standards Committee (Japan), Dirección General de Normas (Mexico), State Committee of the Russian Federation for Standardization and Metrology (Russia), Swiss Association for Standardization (Switzerland), and British Standards Institution (United Kingdom).

contain only the digits 0 and 1 rather than the traditional digits 0 through 9. The odometer starts with a reading of all 0s, and as the car is driven for the first few miles, the rightmost wheel rotates from a 0 to a 1. Then, as that 1 rotates back to a 0, it causes a 1 to appear to its left, producing the pattern 10. The 0 on the right then rotates to a 1, producing 11. Now the rightmost wheel rotates from 1 back to 0, causing the 1 to its left to rotate to a 0 as well. This in turn causes another 1 to appear in the third column, producing the pattern 100. In short, as we drive the car we see the following sequence of odometer readings:

0000
0001
0010
0011
0100
0101
0110
0111
1000

This sequence consists of the binary representations of the integers zero through eight. Although tedious, we could extend this counting technique to discover that the bit pattern consisting of 16 1s represents the value 65535, which confirms our claim that any integer in the range from 0 to 65535 can be encoded using 16 bits.

Due to this efficiency, it is common to store numeric information in a form of binary notation rather than in encoded symbols. We say “a form of binary notation” because the straightforward binary system just described is only the basis for several numeric storage techniques used within machines. Some of these variations of the binary system are discussed later in this chapter. For now, we merely note that a system called **two’s complement** notation (see Section 1.6) is common for storing whole numbers because it provides a convenient method for representing negative numbers as well as positive. For representing numbers

with fractional parts such as 4-1/2 or 3/4, another technique, called **floating-point notation** (see Section 1.7), is used.

Representing Images

One means of representing an image is to interpret the image as a collection of dots, each of which is called a **pixel**, short for “picture element.” The appearance of each pixel is then encoded and the entire image is represented as a collection of these encoded pixels. Such a collection is called a **bit map**. This approach is popular because many display devices, such as printers and display screens, operate on the pixel concept. In turn, images in bit map form are easily formatted for display.

The method of encoding the pixels in a bit map varies among applications. In the case of a simple black-and-white image, each pixel can be represented by a single bit whose value depends on whether the corresponding pixel is black or white. This is the approach used by most facsimile machines. For more elaborate black-and-white photographs, each pixel can be represented by a collection of bits (usually eight), which allows a variety of shades of grayness to be represented. In the case of color images, each pixel is encoded by more complex system. Two approaches are common. In one, which we will call RGB encoding, each pixel is represented as three color components—a red component, a green component, and a blue component—corresponding to the three primary colors of light. One byte is normally used to represent the intensity of each color component. In turn, three bytes of storage are required to represent a single pixel in the original image.

An alternative to simple RGB encoding is to use a “brightness” component and two color components. In this case the “brightness” component, which is called the pixel’s luminance, is essentially the sum of the red, green, and blue components. (Actually, it is considered to be the amount of white light in the pixel, but these details need not concern us here.) The other two components, called the blue chrominance and the red chrominance, are determined by computing the difference between the pixel’s luminance and the amount of blue or red light, respectively, in the pixel. Together these three components contain the information required to reproduce the pixel.

The popularity of encoding images using luminance and chrominance components originated in the field of color television broadcast because this approach provided a means of encoding color images that was also compatible with older black-and-white television receivers. Indeed, a gray-scale version of an image can be produced by using only the luminance components of the encoded color image.

ISO—The International Organization for Standardization

The International Organization for Standardization (more commonly called ISO) was established in 1947 as a worldwide federation of standardization bodies, one from each country. Today, it is headquartered in Geneva, Switzerland, and has more than 100 member bodies as well as numerous correspondent members. (A correspondent member is usually a standardization body from a country that does not have a nationally recognized standardization body. Such members cannot participate directly in the development of standards but are kept informed of ISO activities.) ISO maintains a website at <http://www.iso.org>.

A disadvantage of representing images as bit maps is that an image cannot be rescaled easily to any arbitrary size. Essentially, the only way to enlarge the image is to make the pixels bigger, which leads to a grainy appearance. (This is the technique called “digital zoom” used in digital cameras as opposed to “optical zoom” that is obtained by adjusting the camera lens.)

An alternate way of representing images that avoids this scaling problem is to describe the image as a collection of geometric structures, such as lines and curves, that can be encoded using techniques of analytic geometry. Such a description allows the device that ultimately displays the image to decide how the geometric structures should be displayed rather than insisting that the device reproduce a particular pixel pattern. This is the approach used to produce the scalable fonts that are available via today’s word processing systems. For example, TrueType (developed by Microsoft and Apple) is a system for geometrically describing text symbols. Likewise, PostScript (developed by Adobe Systems) provides a means of describing characters as well as more general pictorial data. This geometric means of representing images is also popular in **computer-aided design (CAD)** systems in which drawings of three-dimensional objects are displayed and manipulated on computer display screens.

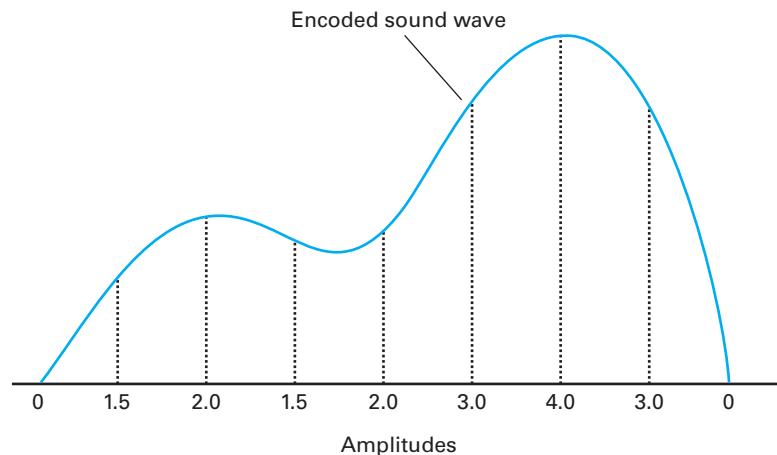
The distinction between representing an image in the form of geometric structures as opposed to bit maps is evident to users of many drawing software systems (such as Microsoft’s Paint utility) that allow the user to draw pictures consisting of pre-established shapes such as rectangles, ovals, and elementary curves. The user simply selects the desired geometric shape from a menu and then directs the drawing of that shape via a mouse. During the drawing process, the software maintains a geometric description of the shape being drawn. As directions are given by the mouse, the internal geometric representation is modified, reconverted to bit map form, and displayed. This allows for easy scaling and shaping of the image. Once the drawing process is complete, however, the underlying geometric description is discarded and only the bit map is preserved, meaning that additional alterations require a tedious pixel-by-pixel modification process. On the other hand, some drawing systems preserve the description as geometric shapes that can be modified later. With these systems, the shapes can be easily resized, maintaining a crisp display at any dimension.

Representing Sound

The most generic method of encoding audio information for computer storage and manipulation is to sample the amplitude of the sound wave at regular intervals and record the series of values obtained. For instance, the series 0, 1.5, 2.0, 1.5, 2.0, 3.0, 4.0, 3.0, 0 would represent a sound wave that rises in amplitude, falls briefly, rises to a higher level, and then drops back to 0 (Figure 1.12). This technique, using a sample rate of 8000 samples per second, has been used for years in long-distance voice telephone communication. The voice at one end of the communication is encoded as numeric values representing the amplitude of the voice every eight-thousandth of a second. These numeric values are then transmitted over the communication line to the receiving end, where they are used to reproduce the sound of the voice.

Although 8000 samples per second may seem to be a rapid rate, it is not sufficient for high-fidelity music recordings. To obtain the quality sound reproduction obtained by today’s musical CDs, a sample rate of 44,100 samples per second

Figure 1.12 The sound wave represented by the sequence 0, 1.5, 2.0, 1.5, 2.0, 3.0, 4.0, 3.0, 0



is used. The data obtained from each sample are represented in 16 bits (32 bits for stereo recordings). Consequently, each second of music recorded in stereo requires more than a million bits.

An alternative encoding system known as Musical Instrument Digital Interface (MIDI, pronounced “MID-ee”) is widely used in the music synthesizers found in electronic keyboards, for video game sound, and for sound effects accompanying websites. By encoding directions for producing music on a synthesizer rather than encoding the sound itself, MIDI avoids the large storage requirements of the sampling technique. More precisely, MIDI encodes what instrument is to play which note for what duration of time, which means that a clarinet playing the note D for two seconds can be encoded in three bytes rather than more than two million bits when sampled at a rate of 44,100 samples per second.

In short, MIDI can be thought of as a way of encoding the sheet music read by a performer rather than the performance itself, and in turn, a MIDI “recording” can sound significantly different when performed on different synthesizers.

Questions & Exercises

1. Here is a message encoded in ASCII using 8 bits per symbol. What does it say? (See Appendix A)
01000011 01101111 01101101 01110000 01110101 01110100 01100101
01110010 00100000 01010011 01100011 01101001 01100101 01101110
01100011 01100101
2. In the ASCII code, what is the relationship between the codes for an uppercase letter and the same letter in lowercase? (See Appendix A.)

3. Encode these sentences in ASCII:
 - a. "Stop!" Cheryl shouted.
 - b. Does $2 + 3 = 5$?
4. Describe a device from everyday life that can be in either of two states, such as a flag on a flagpole that is either up or down. Assign the symbol 1 to one of the states and 0 to the other, and show how the ASCII representation for the letter b would appear when stored with such bits.
5. Convert each of the following binary representations to its equivalent base 10 form:

| | | |
|---------|----------|----------|
| a. 0101 | b. 1001 | c. 1011 |
| d. 0110 | e. 10000 | f. 10010 |
6. Convert each of the following base 10 representations to its equivalent binary form:

| | | |
|-------|-------|-------|
| a. 6 | b. 13 | c. 11 |
| d. 18 | e. 27 | f. 4 |
7. What is the largest numeric value that could be represented with three bytes if each digit were encoded using one ASCII pattern per byte? What if binary notation were used?
8. An alternative to hexadecimal notation for representing bit patterns is **dotted decimal notation** in which each byte in the pattern is represented by its base 10 equivalent. In turn, these byte representations are separated by periods. For example, 12.5 represents the pattern 0000110000000101 (the byte 00001100 is represented by 12, and 00000101 is represented by 5), and the pattern 10001000001000000000111 is represented by 136.16.7. Represent each of the following bit patterns in dotted decimal notation.

| | |
|---------------------|-----------------------------|
| a. 000011100001111 | b. 001100110000000010000000 |
| c. 0000101010100000 | |
9. What is an advantage of representing images via geometric structures as opposed to bit maps? What about bit map techniques as opposed to geometric structures?
10. Suppose a stereo recording of one hour of music is encoded using a sample rate of 44,100 samples per second as discussed in the text. How does the size of the encoded version compare to the storage capacity of a CD?

1.5 The Binary System

In Section 1.4 we saw that binary notation is a means of representing numeric values using only the digits 0 and 1 rather than the 10 digits 0 through 9 that are used in the more common base 10 notational system. It is time now to look at binary notation more thoroughly.

Binary Notation

Recall that in the base 10 system, each position in a representation is associated with a quantity. In the representation 375, the 5 is in the position associated with the quantity one, the 7 is in the position associated with ten, and the 3 is in the position associated with the quantity one hundred (Figure 1.13a). Each quantity is 10 times that of the quantity to its right. The value represented by the entire expression is obtained by multiplying the value of each digit by the quantity associated with that digit's position and then adding those products. To illustrate, the pattern 375 represents $(3 \times \text{hundred}) + (7 \times \text{ten}) + (5 \times \text{one})$, which, in more technical notation, is $(3 \times 10^2) + (7 \times 10^1) + (5 \times 10^0)$.

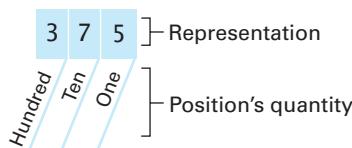
The position of each digit in binary notation is also associated with a quantity, except that the quantity associated with each position is twice the quantity associated with the position to its right. More precisely, the rightmost digit in a binary representation is associated with the quantity one (2^0), the next position to the left is associated with two (2^1), the next is associated with four (2^2), the next with eight (2^3), and so on. For example, in the binary representation 1011, the rightmost 1 is in the position associated with the quantity one, the 1 next to it is in the position associated with two, the 0 is in the position associated with four, and the leftmost 1 is in the position associated with eight (Figure 1.13b).

To extract the value represented by a binary representation, we follow the same procedure as in base 10—we multiply the value of each digit by the quantity associated with its position and add the results. For example, the value represented by 100101 is 37, as shown in Figure 1.14. Note that since binary notation uses only the digits 0 and 1, this multiply-and-add process reduces merely to adding the quantities associated with the positions occupied by 1s. Thus the binary pattern 1011 represents the value eleven, because the 1s are found in the positions associated with the quantities one, two, and eight.

In Section 1.4 we learned how to count in binary notation, which allowed us to encode small integers. For finding binary representations of large values, you may prefer the approach described by the algorithm in Figure 1.15. Let us apply this algorithm to the value thirteen (Figure 1.16). We first divide thirteen by two, obtaining a quotient of six and a remainder of one. Since the quotient was not zero, Step 2 tells us to divide the quotient (six) by two, obtaining a new quotient of three and a remainder of zero. The newest quotient is still not zero, so we divide it by two, obtaining a quotient of one and a remainder of one. Once again, we divide the newest quotient (one) by two, this time obtaining a quotient of zero and a remainder of one. Since we have now acquired a quotient of zero, we move on to Step 3, where we learn that the binary representation of the original value (thirteen) is 1101, obtained from the list of remainders.

Figure 1.13 The base 10 and binary systems

a. Base 10 system



b. Base two system

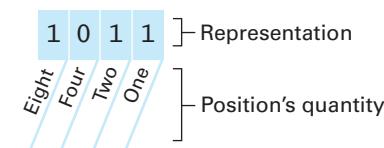
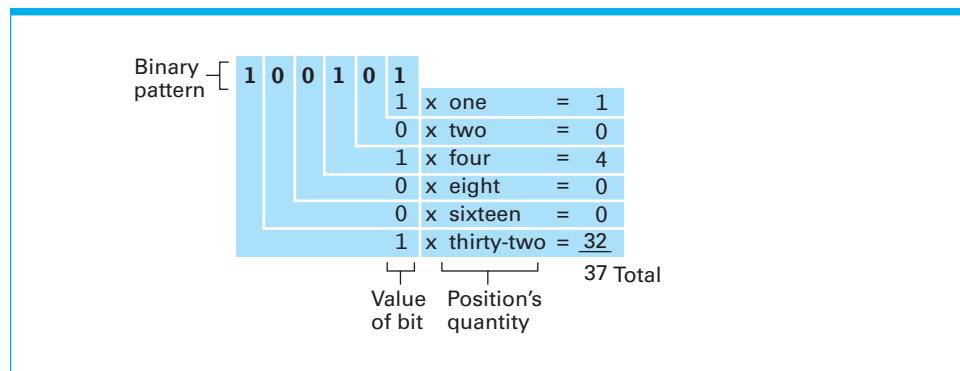
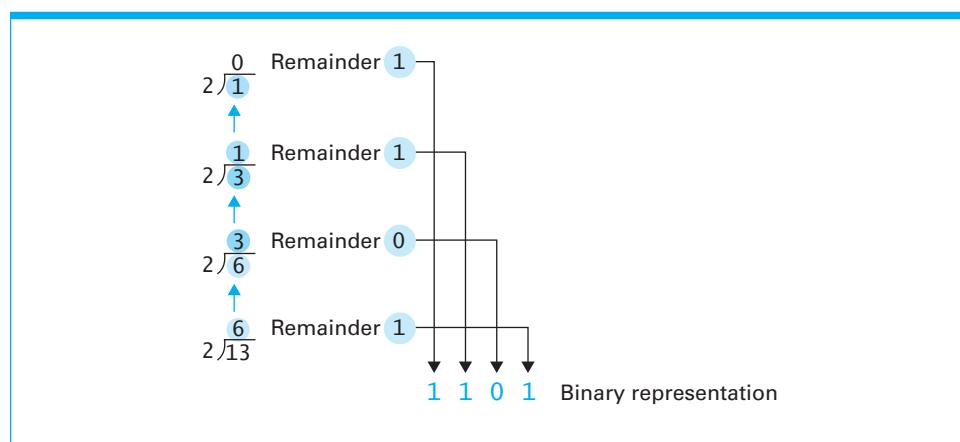


Figure 1.14 Decoding the binary representation 100101**Figure 1.15** An algorithm for finding the binary representation of a positive integer

- Step 1. Divide the value by two and record the remainder.
- Step 2. As long as the quotient obtained is not zero, continue to divide the newest quotient by two and record the remainder.
- Step 3. Now that a quotient of zero has been obtained, the binary representation of the original value consists of the remainders listed from right to left in the order they were recorded.

Figure 1.16 Applying the algorithm in Figure 1.15 to obtain the binary representation of thirteen

Binary Addition

To understand the process of adding two integers that are represented in binary, let us first recall the process of adding values that are represented in traditional base 10 notation. Consider, for example, the following problem:

$$\begin{array}{r} 58 \\ + 27 \\ \hline \end{array}$$

We begin by adding the 8 and the 7 in the rightmost column to obtain the sum 15. We record the 5 at the bottom of that column and carry the 1 to the next column, producing

$$\begin{array}{r} 1 \\ 58 \\ + 27 \\ \hline 5 \end{array}$$

We now add the 5 and 2 in the next column along with the 1 that was carried to obtain the sum 8, which we record at the bottom of the column. The result is as follows:

$$\begin{array}{r} 58 \\ + 27 \\ \hline 85 \end{array}$$

In short, the procedure is to progress from right to left as we add the digits in each column, write the least significant digit of that sum under the column, and carry the more significant digit of the sum (if there is one) to the next column.

To add two integers represented in binary notation, we follow the same procedure except that all sums are computed using the addition facts shown in Figure 1.17 rather than the traditional base 10 facts that you learned in elementary school. For example, to solve the problem

$$\begin{array}{r} 111010 \\ + 11011 \\ \hline \end{array}$$

we begin by adding the rightmost 0 and 1; we obtain 1, which we write below the column. Now we add the 1 and 1 from the next column, obtaining 10. We write the 0 from this 10 under the column and carry the 1 to the top of the next column. At this point, our solution looks like this:

$$\begin{array}{r} 1 \\ 111010 \\ + 11011 \\ \hline 01 \end{array}$$

We add the 1, 0, and 0 in the next column, obtain 1, and write the 1 under this column. The 1 and 1 from the next column total 10; we write the 0 under the column and carry the 1 to the next column. Now our solution looks like this:

$$\begin{array}{r} 1 \\ 111010 \\ + 11011 \\ \hline 0101 \end{array}$$

Figure 1.17 The binary addition facts

| | | | |
|----|----|----|----|
| 0 | 1 | 0 | 1 |
| +0 | +0 | +1 | +1 |
| 0 | 1 | 1 | 10 |

The 1, 1, and 1 in the next column total 11 (binary notation for the value three); we write the low-order 1 under the column and carry the other 1 to the top of the next column. We add that 1 to the 1 already in that column to obtain 10. Again, we record the low-order 0 and carry the 1 to the next column. We now have

$$\begin{array}{r}
 1 \\
 111010 \\
 + 11011 \\
 \hline
 010101
 \end{array}$$

The only entry in the next column is the 1 that we carried from the previous column so we record it in the answer. Our final solution is this:

$$\begin{array}{r}
 111010 \\
 + 11011 \\
 \hline
 1010101
 \end{array}$$

Fractions in Binary

To extend binary notation to accommodate fractional values, we use a **radix point** in the same role as the decimal point in decimal notation. That is, the digits to the left of the point represent the integer part (whole part) of the value and are interpreted as in the binary system discussed previously. The digits to its right represent the fractional part of the value and are interpreted in a manner similar to the other bits, except their positions are assigned fractional quantities. That is, the first position to the right of the radix is assigned the quantity $1/2$ (which is 2^{-1}), the next position the quantity $1/4$ (which is 2^{-2}), the next $1/8$ (which is 2^{-3}), and so on. Note that this is merely a continuation of the rule stated previously: Each position is assigned a quantity twice the size of the one to its right. With these quantities assigned to the bit positions, decoding a binary representation containing a radix point requires the same procedure as used without a radix point. More precisely, we multiply each bit value by the quantity assigned to that bit's position in the representation. To illustrate, the binary representation 101.101 decodes to $5-5/8$, as shown in Figure 1.18.

For addition, the techniques applied in the base 10 system are also applicable in binary. That is, to add two binary representations having radix points, we

Figure 1.18 Decoding the binary representation 101.101

Analog versus Digital

Prior to the twenty-first century, many researchers debated the pros and cons of digital versus analog technology. In a digital system, a value is encoded as a series of digits and then stored using several devices, each representing one of the digits. In an analog system, each value is stored in a single device that can represent any value within a continuous range.

Let us compare the two approaches using buckets of water as the storage devices. To simulate a digital system, we could agree to let an empty bucket represent the digit 0 and a full bucket represent the digit 1. Then we could store a numeric value in a row of buckets using floating-point notation (see Section 1.7). In contrast, we could simulate an analog system by partially filling a single bucket to the point at which the water level represented the numeric value being represented. At first glance, the analog system may appear to be more accurate since it would not suffer from the truncation errors inherent in the digital system (again see Section 1.7). However, any movement of the bucket in the analog system could cause errors in detecting the water level, whereas a significant amount of sloshing would have to occur in the digital system before the distinction between a full bucket and an empty bucket would be blurred. Thus the digital system would be less sensitive to error than the analog system. This robustness is a major reason why many applications that were originally based on analog technology (such as telephone communication, audio recordings, and television) are shifting to digital technology.

merely align the radix points and apply the same addition process as before. For example, 10.011 added to 100.11 produces 111.001, as shown here:

$$\begin{array}{r} 10.011 \\ + 100.110 \\ \hline 111.001 \end{array}$$

Questions & Exercises

1. Convert each of the following binary representations to its equivalent base 10 form:
 - a. 101010
 - b. 100001
 - c. 10111
 - d. 0110
 - e. 11111
2. Convert each of the following base 10 representations to its equivalent binary form:
 - a. 32
 - b. 64
 - c. 96
 - d. 15
 - e. 27
3. Convert each of the following binary representations to its equivalent base 10 form:
 - a. 11.01
 - b. 101.111
 - c. 10.1
 - d. 110.011
 - e. 0.101
4. Express the following values in binary notation:
 - a. $4\frac{1}{2}$
 - b. $2\frac{3}{4}$
 - c. $1\frac{1}{8}$
 - d. $\frac{5}{16}$
 - e. $5\frac{5}{8}$

5. Perform the following additions in binary notation:

| | | | |
|----------|-------------|----------|-----------|
| a. 11011 | b. 1010.001 | c. 11111 | d. 111.11 |
| +1100 | + 1.101 | + 0001 | + 00.01 |

1.6 Storing Integers

Mathematicians have long been interested in numeric notational systems, and many of their ideas have turned out to be very compatible with the design of digital circuitry. In this section we consider two of these notational systems, two's complement notation and excess notation, which are used for representing integer values in computing equipment. These systems are based on the binary system but have additional properties that make them more compatible with computer design. With these advantages, however, come disadvantages as well. Our goal is to understand these properties and how they affect computer usage.

Two's Complement Notation

The most popular system for representing integers within today's computers is **two's complement** notation. This system uses a fixed number of bits to represent each of the values in the system. In today's equipment, it is common to use a two's complement system in which each value is represented by a pattern of 32 bits. Such a large system allows a wide range of numbers to be represented but is awkward for demonstration purposes. Thus, to study the properties of two's complement systems, we will concentrate on smaller systems.

Figure 1.19 shows two complete two's complement systems—one based on bit patterns of length three, the other based on bit patterns of length four. Such a system is constructed by starting with a string of 0s of the appropriate length and then counting in binary until the pattern consisting of a single 0 followed by 1s is reached. These patterns represent the values 0, 1, 2, 3, The patterns representing negative values are obtained by starting with a string of 1s of the appropriate length and then counting backward in binary until the pattern consisting of a single 1 followed by 0s is reached. These patterns represent the values $-1, -2, -3, \dots$. (If counting backward in binary is difficult for you, merely start at the very bottom of the table with the pattern consisting of a single 1 followed by 0s, and count up to the pattern consisting of all 1s.)

Note that in a two's complement system, the leftmost bit of a bit pattern indicates the sign of the value represented. Thus, the leftmost bit is often called the **sign bit**. In a two's complement system, negative values are represented by the patterns whose sign bits are 1; nonnegative values are represented by patterns whose sign bits are 0.

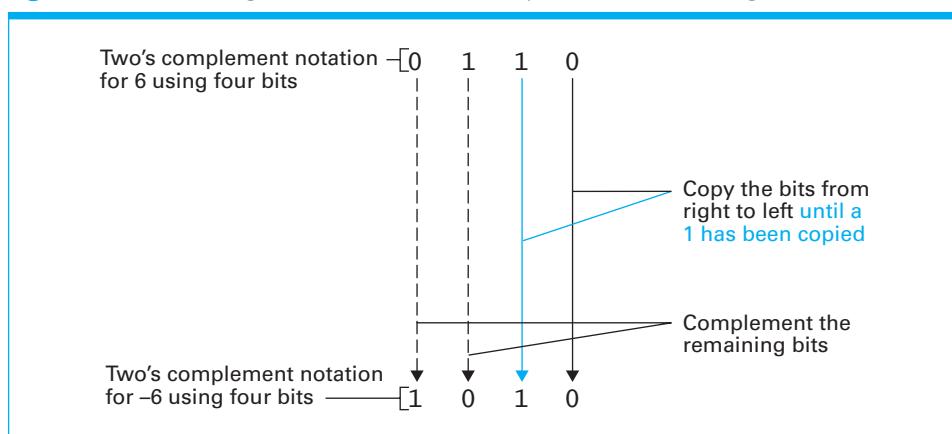
In a two's complement system, there is a convenient relationship between the patterns representing positive and negative values of the same magnitude. They are identical when read from right to left, up to and including the first 1. From there on, the patterns are complements of one another. (The **complement** of a pattern is the pattern obtained by changing all the 0s to 1s and all the 1s to 0s; 0110 and 1001 are complements.) For example, in the 4-bit system in Figure 1.19

Figure 1.19 Two's complement notation systems

| a. Using patterns of length three | | b. Using patterns of length four | |
|-----------------------------------|-------------------|----------------------------------|-------------------|
| Bit pattern | Value represented | Bit pattern | Value represented |
| 011 | 3 | 0111 | 7 |
| 010 | 2 | 0110 | 6 |
| 001 | 1 | 0101 | 5 |
| 000 | 0 | 0100 | 4 |
| 111 | -1 | 0011 | 3 |
| 110 | -2 | 0010 | 2 |
| 101 | -3 | 0001 | 1 |
| 100 | -4 | 0000 | 0 |
| | | 1111 | -1 |
| | | 1110 | -2 |
| | | 1101 | -3 |
| | | 1100 | -4 |
| | | 1011 | -5 |
| | | 1010 | -6 |
| | | 1001 | -7 |
| | | 1000 | -8 |

the patterns representing 2 and -2 both end with 10, but the pattern representing 2 begins with 00, whereas the pattern representing -2 begins with 11. This observation leads to an algorithm for converting back and forth between bit patterns representing positive and negative values of the same magnitude. We merely copy the original pattern from right to left until a 1 has been copied, then we complement the remaining bits as they are transferred to the final bit pattern (Figure 1.20).

Understanding these basic properties of two's complement systems also leads to an algorithm for decoding two's complement representations. If the pattern

Figure 1.20 Encoding the value -6 in two's complement notation using 4 bits

to be decoded has a sign bit of 0, we need merely read the value as though the pattern were a binary representation. For example, 0110 represents the value 6, because 110 is binary for 6. If the pattern to be decoded has a sign bit of 1, we know the value represented is negative, and all that remains is to find the magnitude of the value. We do this by applying the “copy and complement” procedure in Figure 1.20 and then decoding the pattern obtained as though it were a straightforward binary representation. For example, to decode the pattern 1010, we first recognize that since the sign bit is 1, the value represented is negative. Hence, we apply the “copy and complement” procedure to obtain the pattern 0110, recognize that this is the binary representation for 6, and conclude that the original pattern represents -6.

Addition in Two's Complement Notation To add values represented in two's complement notation, we apply the same algorithm that we used for binary addition, except that all bit patterns, including the answer, are the same length. This means that when adding in a two's complement system, any extra bit generated on the left of the answer by a final carry must be truncated. Thus “adding” 0101 and 0010 produces 0111, and “adding” 0111 and 1011 results in 0010 ($0111 + 1011 = 10010$, which is truncated to 0010).

With this understanding, consider the three addition problems in Figure 1.21. In each case, we have translated the problem into two's complement notation (using bit patterns of length four), performed the addition process previously described, and decoded the result back into our usual base 10 notation.

Observe that the third problem in Figure 1.21 involves the addition of a positive number to a negative number, which demonstrates a major benefit of two's complement notation: Addition of any combination of signed numbers can be accomplished using the same algorithm and thus the same circuitry. This is in stark contrast to how humans traditionally perform arithmetic computations. Whereas elementary school children are first taught to add and later taught to subtract, a machine using two's complement notation needs to know only how to add.

Figure 1.21 Addition problems converted to two's complement notation

| Problem in base 10 | Problem in two's complement | Answer in base 10 |
|---|--|-------------------|
| $\begin{array}{r} 3 \\ + 2 \\ \hline \end{array}$ | $\begin{array}{r} 0011 \\ + 0010 \\ \hline 0101 \end{array}$ | 5 |
| $\begin{array}{r} -3 \\ + -2 \\ \hline \end{array}$ | $\begin{array}{r} 1101 \\ + 1110 \\ \hline 1011 \end{array}$ | -5 |
| $\begin{array}{r} 7 \\ + -5 \\ \hline \end{array}$ | $\begin{array}{r} 0111 \\ + 1011 \\ \hline 0010 \end{array}$ | 2 |

For example, the subtraction problem $7 - 5$ is the same as the addition problem $7 + (-5)$. Consequently, if a machine were asked to subtract 5 (stored as 0101) from 7 (stored as 0111), it would first change the 5 to -5 (represented as 1011) and then perform the addition process of $0111 + 1011$ to obtain 0010, which represents 2, as follows:

$$\begin{array}{r} 7 \\ \underline{-5} \\ \rightarrow \quad \underline{-0101} \end{array} \quad \begin{array}{r} 0111 \\ + 1011 \\ \hline 0010 \end{array} \quad \begin{array}{r} \rightarrow \quad 2 \end{array}$$

We see, then, that when two's complement notation is used to represent numeric values, a circuit for addition combined with a circuit for negating a value is sufficient for solving both addition and subtraction problems. (Such circuits are shown and explained in Appendix B.)

The Problem of Overflow One problem we have avoided in the preceding examples is that in any two's complement system there is a limit to the size of the values that can be represented. When using two's complement with patterns of 4 bits, the largest positive integer that can be represented is 7, and the most negative integer is -8 . In particular, the value 9 cannot be represented, which means that we cannot hope to obtain the correct answer to the problem $5 + 4$. In fact, the result would appear as -7 . This phenomenon is called **overflow**. That is, overflow is the problem that occurs when a computation produces a value that falls outside the range of values that can be represented. When using two's complement notation, this might occur when adding two positive values or when adding two negative values. In either case, the condition can be detected by checking the sign bit of the answer. An overflow is indicated if the addition of two positive values results in the pattern for a negative value or if the sum of two negative values appears to be positive.

Of course, because most computers use two's complement systems with longer bit patterns than we have used in our examples, larger values can be manipulated without causing an overflow. Today, it is common to use patterns of 32 bits for storing values in two's complement notation, allowing for positive values as large as 2,147,483,647 to accumulate before overflow occurs. If still larger values are needed, longer bit patterns can be used or perhaps the units of measure can be changed. For instance, finding a solution in terms of miles instead of inches results in smaller numbers being used and might still provide the accuracy required.

The point is that computers can make mistakes. So, the person using the machine must be aware of the dangers involved. One problem is that computer programmers and users become complacent and ignore the fact that small values can accumulate to produce large numbers. For example, in the past it was common to use patterns of 16 bits for representing values in two's complement notation, which meant that overflow would occur when values of $2^{15} = 32,768$ or larger were reached. On September 19, 1989, a hospital computer system malfunctioned after years of reliable service. Close inspection revealed that this date was 32,768 days after January 1, 1900, and the machine was programmed to compute dates based on that starting date. Thus, because of overflow, September 19, 1989, produced a negative value—a phenomenon the computer's program was not designed to handle.

Excess Notation

Another method of representing integer values is **excess notation**. As is the case with two's complement notation, each of the values in an excess notation system is represented by a bit pattern of the same length. To establish an excess system, we first select the pattern length to be used, then write down all the different bit patterns of that length in the order they would appear if we were counting in binary. Next, we observe that the first pattern with a 1 as its most significant bit appears approximately halfway through the list. We pick this pattern to represent zero; the patterns following this are used to represent 1, 2, 3, . . . ; and the patterns preceding it are used for $-1, -2, -3, \dots$. The resulting code, when using patterns of length four, is shown in Figure 1.22. There we see that the value 5 is represented by the pattern 1101 and -5 is represented by 0011. (Note that one difference between an excess system and a two's complement system is that the sign bits are reversed.)

The system represented in Figure 1.22 is known as excess eight notation. To understand why, first interpret each of the patterns in the code using the traditional binary system and then compare these results to the values represented in the excess notation. In each case, you will find that the binary interpretation exceeds the excess notation interpretation by the value 8. For example, the pattern 1100 in binary notation represents the value 12, but in our excess system it represents 4; 0000 in binary notation represents 0, but in the excess system it represents negative 8. In a similar manner, an excess system based on patterns of length five would be called excess 16 notation, because the pattern 10000, for instance, would be used to represent zero rather than representing its usual value of 16. Likewise, you may want to confirm that the three-bit excess system would be known as excess four notation (Figure 1.23).

Figure 1.22 An excess eight conversion table

| Bit pattern | Value represented |
|-------------|-------------------|
| 1111 | 7 |
| 1110 | 6 |
| 1101 | 5 |
| 1100 | 4 |
| 1011 | 3 |
| 1010 | 2 |
| 1001 | 1 |
| 1000 | 0 |
| 0111 | -1 |
| 0110 | -2 |
| 0101 | -3 |
| 0100 | -4 |
| 0011 | -5 |
| 0010 | -6 |
| 0001 | -7 |
| 0000 | -8 |

Figure 1.23 An excess notation system using bit patterns of length three

| Bit pattern | Value represented |
|-------------|-------------------|
| 111 | 3 |
| 110 | 2 |
| 101 | 1 |
| 100 | 0 |
| 011 | -1 |
| 010 | -2 |
| 001 | -3 |
| 000 | -4 |

Questions & Exercises

- Convert each of the following two's complement representations to its equivalent base 10 form:
 - 00011
 - 01111
 - 11100
 - 11010
 - 00000
 - 10000
- Convert each of the following base 10 representations to its equivalent two's complement form using patterns of 8 bits:
 - 6
 - 6
 - 17
 - 13
 - 1
 - 0
- Suppose the following bit patterns represent values stored in two's complement notation. Find the two's complement representation of the negative of each value:
 - 00000001
 - 01010101
 - 11111100
 - 11111110
 - 00000000
 - 01111111
- Suppose a machine stores numbers in two's complement notation. What are the largest and smallest numbers that can be stored if the machine uses bit patterns of the following lengths?
 - four
 - six
 - eight
- In the following problems, each bit pattern represents a value stored in two's complement notation. Find the answer to each problem in two's complement notation by performing the addition process described in the text. Then check your work by translating the problem and your answer into base 10 notation.
 - 0101 + 0010
 - 0011 + 0001
 - 0101 + 1010
 - 1110 + 0011
 - 1010 + 1110

6. Solve each of the following problems in two's complement notation, but this time watch for overflow and indicate which answers are incorrect because of this phenomenon.
 - a. 0100 + 0011
 - b. 0101 + 0110
 - c. 1010 + 1010
 - d. 1010 + 0111
 - e. 0111 + 0001
7. Translate each of the following problems from base 10 notation into two's complement notation using bit patterns of length four, then convert each problem to an equivalent addition problem (as a machine might do), and perform the addition. Check your answers by converting them back to base 10 notation.
 - a. 6 – (–1)
 - b. 3 – (–2)
 - c. 4 – 6
 - d. 2 – (–4)
 - e. 1 – 5
8. Can overflow ever occur when values are added in two's complement notation with one value positive and the other negative? Explain your answer.
9. Convert each of the following excess eight representations to its equivalent base 10 form without referring to the table in the text:
 - a. 1110
 - b. 0111
 - c. 1000
 - d. 0010
 - e. 0000
 - f. 1001
10. Convert each of the following base 10 representations to its equivalent excess eight form without referring to the table in the text:
 - a. 5
 - b. –5
 - c. 3
 - d. 0
 - e. 7
 - f. –8
11. Can the value 9 be represented in excess eight notation? What about representing 6 in excess four notation? Explain your answer.

1.7 Storing Fractions

In contrast to the storage of integers, the storage of a value with a fractional part requires that we store not only the pattern of 0s and 1s representing its binary representation but also the position of the radix point. A popular way of doing this is based on scientific notation and is called **floating-point** notation.

Floating-Point Notation

Let us explain floating-point notation with an example using only one byte of storage. Although machines normally use much longer patterns, this 8-bit format is representative of actual systems and serves to demonstrate the important concepts without the clutter of long bit patterns.

We first designate the high-order bit of the byte as the sign bit. Once again, a 0 in the sign bit will mean that the value stored is nonnegative, and a 1 will mean that the value is negative. Next, we divide the remaining 7 bits of the byte into two groups, or fields: the **exponent field** and the **mantissa field**. Let us designate

the 3 bits following the sign bit as the exponent field and the remaining 4 bits as the mantissa field. Figure 1.24 illustrates how the byte is divided.

We can explain the meaning of the fields by considering the following example. Suppose a byte consists of the bit pattern 01101011. Analyzing this pattern with the preceding format, we see that the sign bit is 0, the exponent is 110, and the mantissa is 1011. To decode the byte, we first extract the mantissa and place a radix point on its left side, obtaining

.1011

Next, we extract the contents of the exponent field (110) and interpret it as an integer stored using the 3-bit excess method (see again Figure 1.24). Thus the pattern in the exponent field in our example represents a positive 2. This tells us to move the radix in our solution to the right by 2 bits. (A negative exponent would mean to move the radix to the left.) Consequently, we obtain

10.11

which is the binary representation for $2^{3/4}$. (Recall the representation of binary fractions from Figure 1.18.) Next, we note that the sign bit in our example is 0; the value represented is thus nonnegative. We conclude that the byte 01101011 represents $2^{3/4}$. Had the pattern been 11101011 (which is the same as before except for the sign bit), the value represented would have been $2^{3/4}$.

As another example, consider the byte 00111100. We extract the mantissa to obtain

.1100

and move the radix 1 bit to the left, since the exponent field (011) represents the value -1 . We therefore have

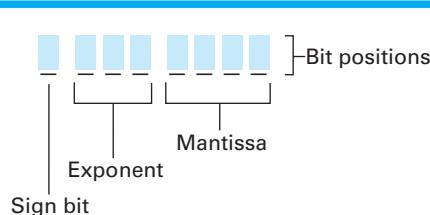
.01100

which represents $3/8$. Since the sign bit in the original pattern is 0, the value stored is nonnegative. We conclude that the pattern 00111100 represents $\frac{3}{8}$.

To store a value using floating-point notation, we reverse the preceding process. For example, to encode $1\frac{1}{8}$, first we express it in binary notation and obtain 1.001. Next, we copy the bit pattern into the mantissa field from left to right, starting with the leftmost 1 in the binary representation. At this point, the byte looks like this:

 1 0 0 1

Figure 1.24 Floating-point notation components



We must now fill in the exponent field. To this end, we imagine the contents of the mantissa field with a radix point at its left and determine the number of bits and the direction the radix must be moved to obtain the original binary number. In our example, we see that the radix in .1001 must be moved 1 bit to the right to obtain 1.001. The exponent should therefore be a positive one, so we place 101 (which is positive one in excess four notation as shown in Figure 1.23) in the exponent field. Finally, we fill the sign bit with 0 because the value being stored is nonnegative. The finished byte looks like this:

0 1 0 1 1 0 0 1

There is a subtle point you may have missed when filling in the mantissa field. The rule is to copy the bit pattern appearing in the binary representation from left to right, starting with the leftmost 1. To clarify, consider the process of storing the value $\frac{3}{8}$, which is .011 in binary notation. In this case the mantissa will be

— — — 1 1 0 0

It will not be

— — — 0 1 1 0

This is because we fill in the mantissa field starting with the leftmost 1 that appears in the binary representation. Representations that conform to this rule are said to be in **normalized form**.

Using normalized form eliminates the possibility of multiple representations for the same value. For example, both 00111100 and 01000110 would decode to the value $\frac{3}{8}$, but only the first pattern is in normalized form. Complying with normalized form also means that the representation for all nonzero values will have a mantissa that starts with 1. The value zero, however, is a special case; its floating-point representation is a bit pattern of all 0s.

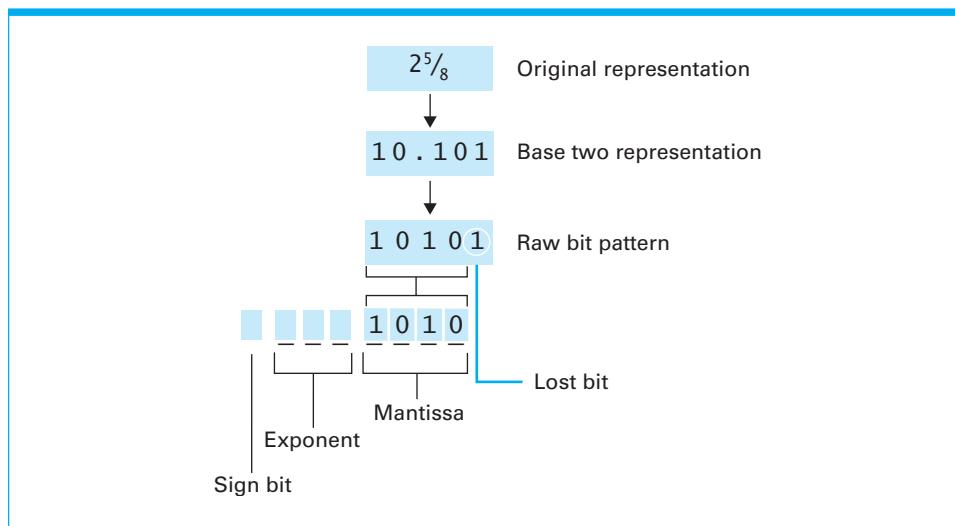
Truncation Errors

Let us consider the annoying problem that occurs if we try to store the value $2^{\frac{5}{8}}$ with our one-byte floating-point system. We first write $2^{\frac{5}{8}}$ in binary, which gives us 10.101. But when we copy this into the mantissa field, we run out of room, and the rightmost 1 (which represents the last $\frac{1}{8}$) is lost (Figure 1.25). If we ignore this problem for now and continue by filling in the exponent field and the sign bit, we end up with the bit pattern 01101010, which represents $2^{\frac{1}{2}}$ instead of $2^{\frac{5}{8}}$. What has occurred is called a **truncation error**, or **round-off error**—meaning that part of the value being stored is lost because the mantissa field is not large enough.

The significance of such errors can be reduced by using a longer mantissa field. In fact, most computers manufactured today use at least 32 bits for storing values in floating-point notation instead of the 8 bits we have used here. This also allows for a longer exponent field at the same time. Even with these longer formats, however, there are still times when more accuracy is required.

Another source of truncation errors is a phenomenon that you are already accustomed to in base 10 notation: the problem of nonterminating expansions, such as those found when trying to express $\frac{1}{3}$ in decimal form. Some values cannot be accurately expressed regardless of how many digits we use. The difference between our traditional base 10 notation and binary notation is that more values have nonterminating representations in binary than in decimal notation. For example, the value $1/10$ is nonterminating when expressed in binary. Imagine the problems this might cause the unwary person using floating-point notation to

Figure 1.25 Encoding the value $2\frac{5}{8}$



store and manipulate dollars and cents. In particular, if the dollar is used as the unit of measure, the value of a dime could not be stored accurately. A solution in this case is to manipulate the data in units of pennies so that all values are integers that can be accurately stored using a method such as two's complement.

Truncation errors and their related problems are an everyday concern for people working in the area of numerical analysis. This branch of mathematics deals with the problems involved when doing actual computations that are often massive and require significant accuracy.

The following is an example that would warm the heart of any numerical analyst. Suppose we are asked to add the following three values using our one-byte floating-point notation defined previously:

$$2\frac{1}{2} + \frac{1}{8} + \frac{1}{8}$$

Single Precision Floating Point

The floating-point notation introduced in this chapter (Section 1.7) is far too simplistic to be used in an actual computer. After all, with just 8 bits, only 256 numbers out of the set of all real numbers can be expressed. Our discussion has used 8 bits to keep the examples simple, yet still cover the important underlying concepts.

Many of today's computers support a 32-bit form of this notation called **Single Precision Floating Point**. This format uses 1 bit for the sign, 8 bits for the exponent (in an excess notation), and 23 bits for the mantissa. Thus, single precision floating point is capable of expressing very large numbers (order of 10^{38}) down to very small numbers (order of 10^{-37}) with the precision of 7 decimal digits. That is to say, the first 7 digits of a given decimal number can be stored with very good accuracy (a small amount of error may still be present). Any digits passed the first 7 will certainly be lost by truncation error (although the magnitude of the number is retained).

Another form, called **Double Precision Floating Point**, uses 64 bits and provides a precision of 15 decimal digits.

If we add the values in the order listed, we first add $2^{1/2}$ to $1/8$ and obtain $2^{5/8}$, which in binary is 10.101. Unfortunately, because this value cannot be stored accurately (as seen previously), the result of our first step ends up being stored as $2^{1/2}$ (which is the same as one of the values we were adding). The next step is to add this result to the last $1/8$. Here again a truncation error occurs, and our final result turns out to be the incorrect answer $2^{1/2}$.

Now let us add the values in the opposite order. We first add $1/8$ to $1/8$ to obtain $1/4$. In binary this is .01; so the result of our first step is stored in a byte as 00111000, which is accurate. We now add this $1/4$ to the next value in the list, $2^{1/2}$, and obtain $2^{3/4}$, which we can accurately store in a byte as 01101011. The result this time is the correct answer.

To summarize, in adding numeric values represented in floating-point notation, the order in which they are added can be important. The problem is that if a very large number is added to a very small number, the small number may be truncated. Thus, the general rule for adding multiple values is to add the smaller values together first, in hopes that they will accumulate to a value that is significant when added to the larger values. This was the phenomenon experienced in the preceding example.

Designers of today's commercial software packages do a good job of shielding the uneducated user from problems such as this. In a typical spreadsheet system, correct answers will be obtained unless the values being added differ in size by a factor of 10^{16} or more. Thus, if you found it necessary to add one to the value

10,000,000,000,000,000

you might get the answer

10,000,000,000,000,000

rather than

10,000,000,000,000,001

Such problems are significant in applications (such as navigational systems) in which minor errors can be compounded in additional computations and ultimately produce significant consequences, but for the typical PC user the degree of accuracy offered by most commercial software is sufficient.

Questions & Exercises

1. Decode the following bit patterns using the floating-point format discussed in the text:
 - a. 01001010
 - b. 01101101
 - c. 00111001
 - d. 11011100
 - e. 10101011
2. Encode the following values into the floating-point format discussed in the text. Indicate the occurrence of truncation errors.
 - a. $2^{3/4}$
 - b. $5^{1/4}$
 - c. $^{3/4}$
 - d. $-3^{1/2}$
 - e. $-4^{3/8}$

3. In terms of the floating-point format discussed in the text, which of the patterns 01001001 and 00111101 represents the larger value? Describe a simple procedure for determining which of two patterns represents the larger value.
4. When using the floating-point format discussed in the text, what is the largest value that can be represented? What is the smallest positive value that can be represented?

1.8 Data and Programming

While humans have devised the data representations and basic operations that comprise modern computers, few people are very good at working with computers directly at this level. People prefer to reason about computational problems at a higher level of abstraction, and they rely on the computer to handle the lowest levels of detail. A *programming language* is a computer system created to allow humans to precisely express algorithms to the computer using a higher level of abstraction.

In the twentieth century, programming computers was considered to be the province of a few highly trained experts; to be sure, there remain many problems in computing that require the attention of experienced computer scientists and software engineers. However, in the twenty-first century, as computers and computing have become increasingly intertwined in every aspect of our modern lives, it has grown steadily more difficult to identify career fields that do not require at least some degree of programming skill. Indeed, some have identified programming or *coding* to be the next foundational pillar of modern literacy, alongside reading, writing, and arithmetic.

In this section, and in programming supplement sections in subsequent chapters, we look at how a programming language reflects the main ideas of the chapter and allows humans to more easily solve problems involving computation.

Getting Started with Python

Python is a programming language that was created by Guido van Rossum in the late 1980s. Today it is one of the top ten most-used languages and remains popular in developing web applications, in scientific computation, and as an introductory language for students. Organizations that use Python range from Google to NASA, DropBox to Industrial Light & Magic, and across the spectrum of casual, scientific, and artistic computer users. Python emphasizes readability and includes elements of the imperative, object-oriented, and functional programming paradigms, which will be explored in Chapter 6.

The software for editing and running programs written in Python is freely available from www.python.org, as are many other resources for getting started. The Python language has evolved and continues to evolve over time. All of the examples in this book will use a version of the language called Python 3. Earlier versions of Python are capable of running very similar programs, but there have been many minor changes, such as punctuation, since Python 2.

Python is an *interpreted language*, which for beginners means that Python instructions can be typed into an interactive prompt or can be stored in a plain text file (called a “script”) and run later. In the examples below, either mode can be used, but exercise and chapter review problems will generally ask for a Python script.

Hello Python

By longstanding tradition, the first program described in many programming language introductions is “Hello, World.” This simple program outputs a nominal greeting, demonstrating how a particular language produces a result, and also how a language represents text. In Python¹, we write this program as

```
print('Hello, World!')
```

Type this statement into Python’s interactive interpreter, or save it as a Python script and execute it. In either case, the result should be:

Hello, World!

Python parrots the text between the quotation marks back to the user.

There are several aspects to note even in this simple Python script. First, `print` is a built-in function, a predefined operation that Python scripts can use to produce *output*, a result of the program that will be made visible to the user. The `print` is followed by opening and closing parentheses; what comes between those parentheses is the value to be printed.

Second, Python can denote strings of text using single quotation marks. The quotation marks in front of the capital H and after the exclamation point denote the beginning and end of a string of characters that will be treated as a value in Python.

Programming languages carry out their instructions very precisely. If a user makes subtle changes to the message between the starting and finishing quotation marks within the `print` statement, the resultant printed text will change accordingly. Take a moment to try different capitalizations, punctuation, and even different words within the `print` statement to see that this is so.

Variables

Python allows the user to name values for later use, an important abstraction when constructing compact, understandable scripts. These named storage locations are termed *variables*, analogous to the mathematical variables often seen in algebra courses. Consider the slightly enhanced version of Hello World below:

```
message = 'Hello, World!'
print(message)
```

In this script, the first line is an *assignment statement*. The use of the `=` can be misleading to beginners, who are accustomed to the algebraic usage of the equal sign. This assignment statement should be read, “variable `message` is assigned

¹This Python code is for version 3 of the language, which will be referred to only as “Python” for the remainder of the book. Earlier versions of Python do not always require the opening and closing parentheses.

the string value '`Hello, World!`'". In general, an assignment statement will have a variable name on the left side of the equal sign and a value to the right.

Python is a *dynamically typed* language, which means that our script need not establish ahead of time that there will be a variable called `message`, or what type of value should be stored in `message`. In the script, it is sufficient to state that our text string will be assigned to `message`, and then to refer to that variable `message` in the subsequent `print` statement.

The naming of variables is largely up to the user in Python. Python's simple rules are that variable names must begin with an alphabet letter and may consist of an arbitrary number of letters, digits, and the underscore character, `_`. While a variable named `m` may be sufficient for a two-line example script, experienced programmers strive to give meaningful, descriptive variable names in their scripts.

Python variable names are *case-sensitive*, meaning that capitalization matters. A variable named `size` is treated as distinct from variables named `Size` or `SIZE`. A small number of *keywords*, names that are reserved for special meaning in Python, cannot be used as variable names. You can view this list by accessing the built-in Python help system.

```
help('keywords')
```

Variables can be used to store all of the types of values that Python is able to represent.

```
my_integer = 5
my_floating_point = 26.2
my_Boolean = True
my_string = 'characters'
```

Observe that the types of values we see here correspond directly to the representations covered earlier in this chapter: Boolean trues and falses (Section 1.1), text (Section 1.4), integers (Section 1.6), and floating point numbers (Section 1.7). With additional Python code (beyond the scope of our simple introduction in this text) we could store image and sound data (Section 1.4) with Python variables, as well.

Python expresses hexadecimal values using a `0x` prefix, as in

```
my_integer = 0xFF
print(my_integer)
```

Specifying a value in hexadecimal does not alter the representation of that value in the computer's memory, which stores integer values as a collection of ones and zeros regardless of the numerical base used in the programmer's reasoning. Hexadecimal notation remains a shortcut for humans, used in situations where that representation may aid in understanding the script. The `print` statement above thus prints 255, the base 10 interpretation of hexadecimal `0xFF`, because that is the default behavior for `print`. More complex adjustments to the `print` statement can be used to output values in other representations, but we confine our discussion here to the more familiar base 10.

Unicode characters, including those beyond the ubiquitous ASCII subset, can be included directly in strings when the text editor supports them,

```
print('₹1000')      # Prints ₹1000, one thousand Indian Rupees
```

or can be specified using four hexadecimal digits following a '\u' prefix.

```
print('\u00A31000')      # Prints £1000, one thousand British
# Pounds Sterling
```

The portion of the string '\u00A3' encodes the Unicode representation of the British pound symbol. The '1000' follows immediately so that there will be no space between the currency symbol and the amount in the final output: £1000.

These example statements introduce another language feature, in addition to Unicode text strings. The # symbol denotes the beginning of a *comment*, a human-readable notation to the Python code that will be ignored by the computer when executed. Experienced programmers use comments in their code to explain difficult segments of the algorithm, include history or authorship information, or just to note where a human should pay attention when reading the code. All of the characters to the right of the # until the end of the line are ignored by Python.

Operators and Expressions

Python's built-in operators allow values to be manipulated and combined in a variety of familiar ways.

```
print(3 + 4)      # Prints "7", which is 3 plus 4.
print(5 - 6)      # Prints "-1", which is 5 minus 6
print(7 * 8)      # Prints "56", which is 7 times 8
print(45 / 4)     # Prints "11.25", which is 45 divided by 4
print(2 ** 10)    # Prints "1024", which is 2 to the 10th power
```

When an operation such as forty-five divided by four produces a non-integer result, such as 11.25, Python implicitly switches to a floating-point representation. When purely integer answers are desired, a different set of operators can be used.

```
print(45 // 4)    # Prints "11", which is 45 integer divided by 4
print(45 % 4)     # Prints "1", because 4 * 11 + 1 = 45
```

The double slash signifies the *integer floor division* operator, while the percentage symbol signifies the *modulus*, or remainder operator. Taken together, we can read these calculations as, "four goes into forty-five eleven times, with a remainder of one." In the earlier example, we used ** to signify exponentiation, which can be somewhat surprising given that the caret symbol, ^, is often used for this purpose in typewritten text and even some other programming languages. In Python, the caret operator belongs to the group of *bitwise Boolean operations*, which will be discussed in the next chapter.

String values also can be combined and manipulated in some intuitive ways.

```
s = 'hello' + 'world'
t = s * 4

print(t)      # Prints "helloworldhelloworldhelloworldhelloworld"
```

The plus operator *concatenates* string values, while the multiplication operator *replicates* string values.

The multiple meanings of some of the built-in operators can lead to confusion. This script will produce an error:

```
print('USD$' + 1000)      # TypeError: Can't convert 'int' to str implicitly
```

The error indicates that the string concatenation operator doesn't know what to do when the second operand is not also a string. Fortunately, Python provides functions that allow values to be converted from one type of representation to another. The `int()` function will convert a floating-point value back to an integer representation, discarding the fractional part. It will also convert a string of text digits into an integer representation, provided that the string correctly spells out a valid number. Likewise, the `str()` function can be used to convert numeric representations into UTF-8 encoded text strings. Thus, the following modification to the `print` statement above corrects the error.

```
print('USD$' + str(1000))      # Prints "USD$1000"
```

Currency Conversion

The complete Python script example below demonstrates many of the concepts introduced in this section. Given a set number of U.S. dollars, the script produces monetary conversions to four other currencies.

```
# A converter for international currency exchange.  
USD_to_GBP = 0.66      # Today's rate, US dollars to British Pounds  
USD_to_EUR = 0.77      # Today's rate, US dollars to Euros  
USD_to_JPY = 99.18     # Today's rate, US dollars to Japanese Yen  
USD_to_INR = 59.52     # Today's rate, US dollars to Indian Rupees  
  
GBP_sign    = '\u00A3' # Unicode values for non-ASCII currency symbols.  
EUR_sign    = '\u20AC'  
JPY_sign    = '\u00A5'  
INR_sign    = '\u20B9'  
  
dollars    = 1000 # The number of dollars to convert  
  
pounds     = dollars * USD_to_GBP      # Conversion calculations  
euros      = dollars * USD_to_EUR  
yen        = dollars * USD_to_JPY  
rupees     = dollars * USD_to_INR  
  
print('Today, $' + str(dollars))    # Printing the results  
print('converts to ' + GBP_sign + str(pounds))
```

```
print('converts to ' + EUR_sign + str(euros))
print('converts to ' + JPY_sign + str(yen))
print('converts to ' + INR_sign + str(rupees))
```

When executed, this script outputs the following:

```
Today, $1000
converts to £660.0
converts to €770.0
converts to ¥99180.0
converts to ₹59520.0
```

Debugging

Programming languages are not very forgiving for beginners, and a great deal of time learning to write software can be spent trying to find **bugs**, or errors in the code. There are three major classes of bug that we create in software: **syntax errors** (mistakes in the symbols that have been typed), **semantic errors** (mistakes in the meaning of the program), and **runtime errors** (mistakes that occur when the program is executed.)

Syntax errors are the most common for novices and include simple errors such as forgetting one of the quote marks at the beginning or ending of a text string, failing to close open parentheses, or misspelling the function name `print`. The Python interpreter will generally try to point these errors out when it encounters them, displaying an offending line number and a description of the problem. With some practice, a beginner can quickly learn to recognize and interpret common error cases. As examples:

```
print(5 + )
SyntaxError: invalid syntax
```

This expression is missing a value between the addition operator and the closing parenthesis.

```
print(5.e)
SyntaxError: invalid token
```

Python expects digits to follow the decimal point, not a letter.

```
pront(5)
NameError: name 'pront' is not defined
```

Like calling someone by the wrong name, misspelling the name of a known function or variable can result in confusion and embarrassment.

Semantic errors are flaws in the algorithm, or flaws in the way the algorithm is expressed in a language. Examples might include using the wrong variable name in a calculation or getting the order of arithmetic operations wrong in a complex expression. Python follows the standard rules for operator precedence, so in an expression like `total_pay = 40 + extra_hours * pay_rate`, the multiplication will be performed before the addition, incorrectly calculating the total pay. (Unless your pay rate happens to be \$1/hour.) Use parenthesis to properly specify the order of operations in complex expressions, thereby avoiding both semantic errors and code that may be harder to understand (e.g., `total_pay = (40 + extra_hours) * pay_rate`).

Finally, runtime errors at this level might include unintentionally dividing by zero or using a variable before you have defined it. Python reads statements from top to bottom; it must see an assignment statement to a variable before that variable is used in an expression.

Testing is an integral part of writing Python scripts—or really any kind of program—effectively. Run your script frequently as you write it, perhaps as often as after you complete each line of code. This allows syntax errors to be identified and fixed early and helps focus the author's attention on what should be happening at each step of the script.

Questions & Exercises

1. What makes Python an *interpreted* programming language?
2. Write Python statements that print the following:
 - a. The words “Computer Science Rocks”, followed by an exclamation point
 - b. The number 42
 - c. An approximation of the value of Pi to 4 decimal places
3. Write Python statements to make the following assignments to variables:
 - a. The word “programmer” to a variable called, `rockstar`
 - b. The number of seconds in an hour to a variable called `seconds_per_hour`
 - c. The average temperature of the human body to a variable called `bodyTemp`
4. Write a Python statement that given an existing variable called `bodyTemp` in degrees Fahrenheit stores the equivalent temperature in degrees Celsius to a new variable called `metricBodyTemp`.

1.9 Data Compression

For the purpose of storing or transferring data, it is often helpful (and sometimes mandatory) to reduce the size of the data involved while retaining the underlying information. The technique for accomplishing this is called **data compression**. We begin this section by considering some generic data compression methods and then look at some approaches designed for specific applications.

Generic Data Compression Techniques

Data compression schemes fall into two categories. Some are **lossless**, others are **lossy**. Lossless schemes are those that do not lose information in the compression process. Lossy schemes are those that may lead to the loss of information. Lossy techniques often provide more compression than lossless ones and are

therefore popular in settings in which minor errors can be tolerated, as in the case of images and audio.

In cases where the data being compressed consist of long sequences of the same value, the compression technique called **run-length encoding**, which is a lossless method, is popular. It is the process of replacing sequences of identical data elements with a code indicating the element that is repeated and the number of times it occurs in the sequence. For example, less space is required to indicate that a bit pattern consists of 253 ones, followed by 118 zeros, followed by 87 ones than to actually list all 458 bits.

Another lossless data compression technique is **frequency-dependent encoding**, a system in which the length of the bit pattern used to represent a data item is inversely related to the frequency of the item's use. Such codes are examples of variable-length codes, meaning that items are represented by patterns of different lengths. David Huffman is credited with discovering an algorithm that is commonly used for developing frequency-dependent codes, and it is common practice to refer to codes developed in this manner as **Huffman codes**. In turn, most frequency-dependent codes in use today are Huffman codes.

As an example of frequency-dependent encoding, consider the task of encoded English language text. In the English language the letters *e*, *t*, *a*, and *i* are used more frequently than the letters *z*, *q*, and *x*. So, when constructing a code for text in the English language, space can be saved by using short bit patterns to represent the former letters and longer bit patterns to represent the latter ones. The result would be a code in which English text would have shorter representations than would be obtained with uniform-length codes.

In some cases, the stream of data to be compressed consists of units, each of which differs only slightly from the preceding one. An example would be consecutive frames of a motion picture. In these cases, techniques using **relative encoding**, also known as **differential encoding**, are helpful. These techniques record the differences between consecutive data units rather than entire units; that is, each unit is encoded in terms of its relationship to the previous unit. Relative encoding can be implemented in either lossless or lossy form depending on whether the differences between consecutive data units are encoded precisely or approximated.

Still other popular compression systems are based on **dictionary encoding** techniques. Here the term *dictionary* refers to a collection of building blocks from which the message being compressed is constructed, and the message itself is encoded as a sequence of references to the dictionary. We normally think of dictionary encoding systems as lossless systems, but as we will see in our discussion of image compression, there are times when the entries in the dictionary are only approximations of the correct data elements, resulting in a lossy compression system.

Dictionary encoding can be used by word processors to compress text documents because the dictionaries already contained in these processors for the purpose of spell checking make excellent compression dictionaries. In particular, an entire word can be encoded as a single reference to this dictionary rather than as a sequence of individual characters encoded using a system such as UTF-8. A typical dictionary in a word processor contains approximately 25,000 entries, which means an individual entry can be identified by an integer in the range of 0 to 24,999. This means that a particular entry in the dictionary can be identified by a pattern of only 15 bits. In contrast, if the word being referenced

consisted of six letters, its character-by-character encoding would require 48 bits using UTF-8.

A variation of dictionary encoding is **adaptive dictionary encoding** (also known as dynamic dictionary encoding). In an adaptive dictionary encoding system, the dictionary is allowed to change during the encoding process. A popular example is **Lempel-Ziv-Welsh (LZW) encoding** (named after its creators, Abraham Lempel, Jacob Ziv, and Terry Welsh). To encode a message using LZW, one starts with a dictionary containing the basic building blocks from which the message is constructed, but as larger units are found in the message, they are added to the dictionary—meaning that future occurrences of those units can be encoded as single, rather than multiple, dictionary references. For example, when encoding English text, one could start with a dictionary containing individual characters, digits, and punctuation marks. But as words in the message are identified, they could be added to the dictionary. Thus, the dictionary would grow as the message is encoded, and as the dictionary grows, more words (or recurring patterns of words) in the message could be encoded as single references to the dictionary.

The result would be a message encoded in terms of a rather large dictionary that is unique to that particular message. But this large dictionary would not have to be present to decode the message. Only the original small dictionary would be needed. Indeed, the decoding process could begin with the same small dictionary with which the encoding process started. Then, as the decoding process continues, it would encounter the same units found during the encoding process, and thus be able to add them to the dictionary for future reference just as in the encoding process.

To clarify, consider applying LZW encoding to the message

xyx xyx xyx xyx

starting with a dictionary with three entries, the first being *x*, the second being *y*, and the third being a space. We would begin by encoding *xyx* as 121, meaning that the message starts with the pattern consisting of the first dictionary entry, followed by the second, followed by the first. Then the space is encoded to produce 1213. But, having reached a space, we know that the preceding string of characters forms a word, and so we add the pattern *xyx* to the dictionary as the fourth entry. Continuing in this manner, the entire message would be encoded as 121343434.

If we were now asked to decode this message, starting with the original three-entry dictionary, we would begin by decoding the initial string 1213 as *xyx* followed by a space. At this point we would recognize that the string *xyx* forms a word and add it to the dictionary as the fourth entry, just as we did during the encoding process. We would then continue decoding the message by recognizing that the 4 in the message refers to this new fourth entry and decode it as the word *xyx*, producing the pattern

xyx xyx

Continuing in this manner we would ultimately decode the string 121343434 as

xyx xyx xyx xyx

which is the original message.

Compressing Images

In Section 1.4, we saw how images are encoded using bit map techniques. Unfortunately, the bit maps produced are often very large. In turn, numerous compression schemes have been developed specifically for image representations.

One system known as **GIF** (short for **Graphic Interchange Format** and pronounced “Giff” by some and “Jiff” by others) is a dictionary encoding system that was developed by CompuServe. It approaches the compression problem by reducing the number of colors that can be assigned to a pixel to only 256. The red-green-blue combination for each of these colors is encoded using three bytes, and these 256 encodings are stored in a table (a dictionary) called the palette. Each pixel in an image can then be represented by a single byte whose value indicates which of the 256 palette entries represents the pixel’s color. (Recall that a single byte can contain any one of 256 different bit patterns.) Note that GIF is a lossy compression system when applied to arbitrary images because the colors in the palette may not be identical to the colors in the original image.

GIF can obtain additional compression by extending this simple dictionary system to an adaptive dictionary system using LZW techniques. In particular, as patterns of pixels are encountered during the encoding process, they are added to the dictionary so that future occurrences of these patterns can be encoded more efficiently. Thus, the final dictionary consists of the original palette and a collection of pixel patterns.

One of the colors in a GIF palette is normally assigned the value “transparent,” which means that the background is allowed to show through each region assigned that “color.” This option, combined with the relative simplicity of the GIF system, makes GIF a logical choice in simple animation applications in which multiple images must move around on a computer screen. On the other hand, its ability to encode only 256 colors renders it unsuitable for applications in which higher precision is required, as in the field of photography.

Another popular compression system for images is **JPEG** (pronounced “JAY-peg”). It is a standard developed by the **Joint Photographic Experts Group** (hence the standard’s name) within ISO. JPEG has proved to be an effective standard for compressing color photographs and is widely used in the photography industry, as witnessed by the fact that most digital cameras use JPEG as their default compression technique.

The JPEG standard actually encompasses several methods of image compression, each with its own goals. In those situations that require the utmost in precision, JPEG provides a lossless mode. However, JPEG’s lossless mode does not produce high levels of compression when compared to other JPEG options. Moreover, other JPEG options have proven very successful, meaning that JPEG’s lossless mode is rarely used. Instead, the option known as JPEG’s baseline standard (also known as JPEG’s lossy sequential mode) has become the standard of choice in many applications.

Image compression using the JPEG baseline standard requires a sequence of steps, some of which are designed to take advantage of a human eye’s limitations. In particular, the human eye is more sensitive to changes in brightness than to changes in color. So, starting from an image that is encoded in terms of luminance and chrominance components, the first step is to average the chrominance values over two-by-two pixel squares. This reduces the size of the chrominance information by a factor of four while preserving all the original brightness information. The result is a significant degree of compression without a noticeable loss of image quality.

The next step is to divide the image into eight-by-eight pixel blocks and to compress the information in each block as a unit. This is done by applying a mathematical technique known as the discrete cosine transform, whose details need not concern us here. The important point is that this transformation converts the original eight-by-eight block into another block whose entries reflect how the pixels in the original block relate to each other rather than the actual pixel values. Within this new block, values below a predetermined threshold are then replaced by zeros, reflecting the fact that the changes represented by these values are too subtle to be detected by the human eye. For example, if the original block contained a checkerboard pattern, the new block might reflect a uniform average color. (A typical eight-by-eight pixel block would represent a very small square within the image so the human eye would not identify the checkerboard appearance anyway.)

At this point, more traditional run-length encoding, relative encoding, and variable-length encoding techniques are applied to obtain additional compression. All together, JPEG's baseline standard normally compresses color images by a factor of at least 10, and often by as much as 30, without noticeable loss of quality.

Still another data compression system associated with images is **TIFF** (short for **Tagged Image File Format**). However, the most popular use of TIFF is not as a means of data compression but instead as a standardized format for storing photographs along with related information such as date, time, and camera settings. In this context, the image itself is normally stored as red, green, and blue pixel components without compression.

The TIFF collection of standards does include data compression techniques, most of which are designed for compressing images of text documents in facsimile applications. These use variations of run-length encoding to take advantage of the fact that text documents consist of long strings of white pixels. The color image compression option included in the TIFF standards is based on techniques similar to those used by GIF and are therefore not widely used in the photography community.

Compressing Audio and Video

The most commonly used standards for encoding and compressing audio and video were developed by the **Motion Picture Experts Group (MPEG)** under the leadership of ISO. In turn, these standards themselves are called MPEG.

MPEG encompasses a variety of standards for different applications. For example, the demands for high definition television (HDTV) broadcast are distinct from those for video conferencing, in which the broadcast signal must find its way over a variety of communication paths that may have limited capabilities. Both of these applications differ from that of storing video in such a manner that sections can be replayed or skipped over.

The techniques employed by MPEG are well beyond the scope of this text, but in general, video compression techniques are based on video being constructed as a sequence of pictures in much the same way that motion pictures are recorded on film. To compress such sequences, only some of the pictures, called I-frames, are encoded in their entirety. The pictures between the I-frames are encoded using relative encoding techniques. That is, rather than encode the entire picture, only its distinctions from the prior image are recorded. The I-frames themselves are usually compressed with techniques similar to JPEG.

The best known system for compressing audio is **MP3**, which was developed within the MPEG standards. In fact, the acronym MP3 is short for

MPEG layer 3. Among other compression techniques, MP3 takes advantage of the properties of the human ear, removing those details that the human ear cannot perceive. One such property, called **temporal masking**, is that for a short period after a loud sound, the human ear cannot detect softer sounds that would otherwise be audible. Another, called **frequency masking**, is that a sound at one frequency tends to mask softer sounds at nearby frequencies. By taking advantage of such characteristics, MP3 can be used to obtain significant compression of audio while maintaining near CD quality sound.

Using MPEG and MP3 compression techniques, video cameras are able to record as much as an hour's worth of video within 128MB of storage, and portable music players can store as many as 400 popular songs in a single GB. But, in contrast to the goals of compression in other settings, the goal of compressing audio and video is not necessarily to save storage space. Just as important is the goal of obtaining encodings that allow information to be transmitted over today's communication systems fast enough to provide timely presentation. If each video frame required a MB of storage and the frames had to be transmitted over a communication path that could relay only one KB per second, there would be no hope of successful video conferencing. Thus, in addition to the quality of reproduction allowed, audio and video compression systems are often judged by the transmission speeds required for timely data communication. These speeds are normally measured in **bits per second (bps)**. Common units include **Kbps** (kilo-bps, equal to one thousand bps), **Mbps** (mega-bps, equal to one million bps), and **Gbps** (giga-bps, equal to one billion bps). Using MPEG techniques, video presentations can be successfully relayed over communication paths that provide transfer rates of 40 Mbps. MP3 recordings generally require transfer rates of no more than 64 Kbps.

Questions & Exercises

1. List four generic compression techniques.
2. What would be the encoded version of the message
xyx yxxxx xyx yxxxx yxxxx
if LZW compression, starting with the dictionary containing *x*, *y*, and a space (as described in the text), were used?
3. Why would GIF be better than JPEG when encoding color cartoons?
4. Suppose you were part of a team designing a spacecraft that will travel to other planets and send back photographs. Would it be a good idea to compress the photographs using GIF or JPEG's baseline standard to reduce the resources required to store and transmit the images?
5. What characteristic of the human eye does JPEG's baseline standard exploit?
6. What characteristic of the human ear does MP3 exploit?
7. Identify a troubling phenomenon that is common when encoding numeric information, images, and sound as bit patterns.

1.10 Communication Errors

When information is transferred back and forth among the various parts of a computer, or transmitted from the earth to the moon and back, or, for that matter, merely left in storage, a chance exists that the bit pattern ultimately retrieved may not be identical to the original one. Particles of dirt or grease on a magnetic recording surface or a malfunctioning circuit may cause data to be incorrectly recorded or read. Static on a transmission path may corrupt portions of the data. In the case of some technologies, normal background radiation can alter patterns stored in a machine's main memory.

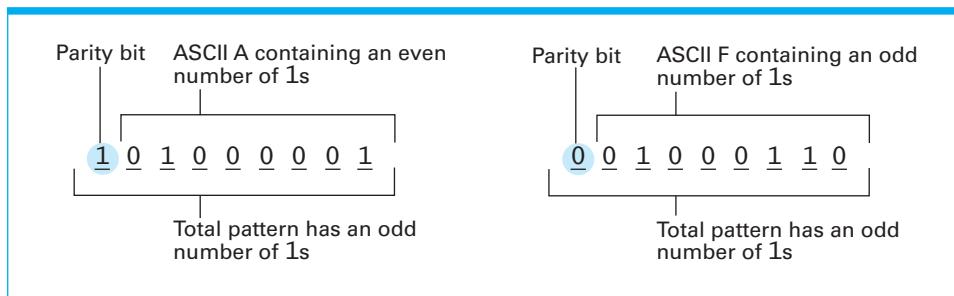
To resolve such problems, a variety of encoding techniques have been developed to allow the detection and even the correction of errors. Today, because these techniques are largely built into the internal components of a computer system, they are not apparent to the personnel using the machine. Nonetheless, their presence is important and represents a significant contribution to scientific research. It is fitting, therefore, that we investigate some of these techniques that lie behind the reliability of today's equipment.

Parity Bits

A simple method of detecting errors is based on the principle that if each bit pattern being manipulated has an odd number of 1s and a pattern with an even number of 1s is encountered, an error must have occurred. To use this principle, we need an encoding system in which each pattern contains an odd number of 1s. This is easily obtained by first adding an additional bit, called a **parity bit**, to each pattern in an encoding system already available (perhaps at the high-order end). In each case, we assign the value 1 or 0 to this new bit so that the entire resulting pattern has an odd number of 1s. Once our encoding system has been modified in this way, a pattern with an even number of 1s indicates that an error has occurred and that the pattern being manipulated is incorrect.

Figure 1.26 demonstrates how parity bits could be added to the ASCII codes for the letters A and F. Note that the code for A becomes 10100001 (parity bit 1) and the ASCII for F becomes 001000110 (parity bit 0). Although the original 8-bit pattern for A has an even number of 1s and the original 8-bit pattern for F has an odd number of 1s, both the 9-bit patterns have an odd number of 1s. If this technique were applied to all the 8-bit ASCII patterns, we would obtain a 9-bit encoding system in which an error would be indicated by any 9-bit pattern with an even number of 1s.

Figure 1.26 The ASCII codes for the letters A and F adjusted for odd parity



The parity system just described is called **odd parity**, because we designed our system so that each correct pattern contains an odd number of 1s. Another technique is called **even parity**. In an even parity system, each pattern is designed to contain an even number of 1s, and thus an error is signaled by the occurrence of a pattern with an odd number of 1s.

Today it is not unusual to find parity bits being used in a computer's main memory. Although we envision these machines as having memory cells of 8-bit capacity, in reality each has a capacity of 9 bits, 1 bit of which is used as a parity bit. Each time an 8-bit pattern is given to the memory circuitry for storage, the circuitry adds a parity bit and stores the resulting 9-bit pattern. When the pattern is later retrieved, the circuitry checks the parity of the 9-bit pattern. If this does not indicate an error, then the memory removes the parity bit and confidently returns the remaining 8-bit pattern. Otherwise, the memory returns the 8 data bits with a warning that the pattern being returned may not be the same pattern that was originally entrusted to memory.

The straightforward use of parity bits is simple, but it has its limitations. If a pattern originally has an odd number of 1s and suffers two errors, it will still have an odd number of 1s, and thus the parity system will not detect the errors. In fact, straightforward applications of parity bits fail to detect any even number of errors within a pattern.

One means of minimizing this problem is sometimes applied to long bit patterns, such as the string of bits recorded in a sector on a magnetic disk. In this case the pattern is accompanied by a collection of parity bits making up a **checkbyte**. Each bit within the checkbyte is a parity bit associated with a particular collection of bits scattered throughout the pattern. For instance, one parity bit may be associated with every eighth bit in the pattern starting with the first bit, while another may be associated with every eighth bit starting with the second bit. In this manner, a collection of errors concentrated in one area of the original pattern is more likely to be detected, since it will be in the scope of several parity bits. Variations of this checkbyte concept lead to error detection schemes known as **checksums** and **cyclic redundancy checks (CRC)**.

Error-Correcting Codes

Although the use of a parity bit allows the detection of an error, it does not provide the information needed to correct the error. Many people are surprised that **error-correcting codes** can be designed so that errors can be not only detected but also corrected. After all, intuition says that we cannot correct errors in a received message unless we already know the information in the message. However, a simple code with such a corrective property is presented in Figure 1.27.

To understand how this code works, we first define the term **Hamming distance**, which is named after R. W. Hamming, who pioneered the search for error-correcting codes after becoming frustrated with the lack of reliability of the early relay machines of the 1940s. The Hamming distance between two bit patterns is the number of bits in which the patterns differ. For example, the Hamming distance between the patterns representing A and B in the code in Figure 1.27 is four, and the Hamming distance between B and C is three. The

Figure 1.27 An error-correcting code

| Symbol | Code |
|--------|--------|
| A | 000000 |
| B | 001111 |
| C | 010011 |
| D | 011100 |
| E | 100110 |
| F | 101001 |
| G | 110101 |
| H | 111010 |

important feature of the code in Figure 1.27 is that any two patterns are separated by a Hamming distance of at least three.

If a single bit is modified in a pattern from Figure 1.27, the error can be detected since the result will not be a legal pattern. (We must change at least 3 bits in any pattern before it will look like another legal pattern.) Moreover, we can also figure out what the original pattern was. After all, the modified pattern will be a Hamming distance of only one from its original form but at least two from any of the other legal patterns.

Thus, to decode a message that was originally encoded using Figure 1.27, we simply compare each received pattern with the patterns in the code until we find one that is within a distance of one from the received pattern. We consider this to be the correct symbol for decoding. For example, if we received the bit pattern 010100 and compared this pattern to the patterns in the code, we would obtain the table in Figure 1.28. Thus, we would conclude that the character transmitted must have been a D because this is the closest match.

Figure 1.28 Decoding the pattern 010100 using the code in Figure 1.27

| Character | Code | Pattern received | Distance between received pattern and code |
|-----------|-------------|------------------|--|
| A | 0 0 0 0 0 0 | 0 1 0 1 0 0 | 2 |
| B | 0 0 1 1 1 1 | 0 1 0 1 0 0 | 4 |
| C | 0 1 0 0 1 1 | 0 1 0 1 0 0 | 3 |
| D | 0 1 1 1 0 0 | 0 1 0 1 0 0 | 1 |
| E | 1 0 0 1 1 0 | 0 1 0 1 0 0 | 3 |
| F | 1 0 1 0 0 1 | 0 1 0 1 0 0 | 5 |
| G | 1 1 0 1 0 1 | 0 1 0 1 0 0 | 2 |
| H | 1 1 1 0 1 0 | 0 1 0 1 0 0 | 4 |

Smallest distance

You will observe that using this technique with the code in Figure 1.27 actually allows us to detect up to two errors per pattern and to correct one error. If we designed the code so that each pattern was a Hamming distance of at least five from each of the others, we would be able to detect up to four errors per pattern and correct up to two. Of course, the design of efficient codes associated with large Hamming distances is not a straightforward task. In fact, it constitutes a part of the branch of mathematics called algebraic coding theory, which is a subject within the fields of linear algebra and matrix theory.

Error-correcting techniques are used extensively to increase the reliability of computing equipment. For example, they are often used in high-capacity magnetic disk drives to reduce the possibility that flaws in the magnetic surface will corrupt data. Moreover, a major distinction between the original CD format used for audio disks and the later format used for computer data storage is in the degree of error correction involved. CD-DA format incorporates error-correcting features that reduce the error rate to only one error for two CDs. This is quite adequate for audio recordings, but a company using CDs to supply software to customers would find that flaws in 50 percent of the disks would be intolerable. Thus, additional error-correcting features are employed in CDs used for data storage, reducing the probability of error to one in 20,000 disks.

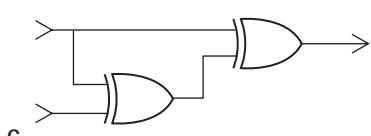
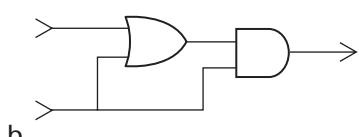
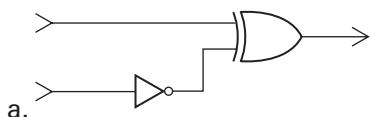
Questions & Exercises

1. The following bytes were originally encoded using odd parity. In which of them do you know that an error has occurred?
a. 100101101 **b.** 100000001 **c.** 000000000
d. 111000000 **e.** 011111111
2. Could errors have occurred in a byte from question 1 without your knowing it? Explain your answer.
3. How would your answers to questions 1 and 2 change if you were told that even parity had been used instead of odd?
4. Encode these sentences in ASCII using odd parity by adding a parity bit at the high-order end of each character code:
a. "Stop!" Cheryl shouted. **b.** Does $2 + 3 = 5$?
5. Using the error-correcting code presented in Figure 1.27, decode the following messages:
a. 001111 100100 001100 **b.** 010001 000000 001011
c. 011010 110110 100000 011100
6. Construct a code for the characters A, B, C, and D using bit patterns of length five so that the Hamming distance between any two patterns is at least three.

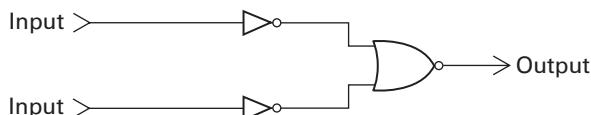
Chapter Review Problems

(Asterisked problems are associated with optional sections.)

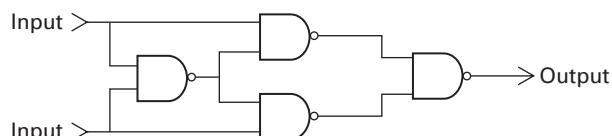
- 1.** Determine the output of each of the following circuits, assuming that the upper input is 1 and the lower input is 0. What would be the output when upper input is 0 and the lower input is 1?



- 2. a.** What Boolean operation does the circuit compute?

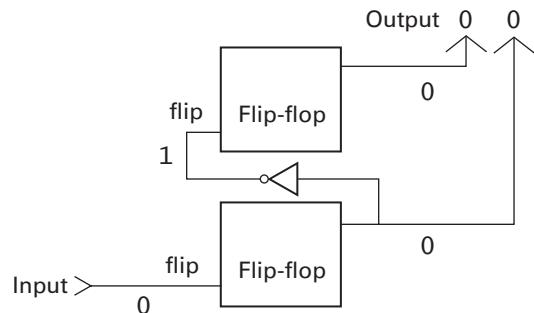


- b.** What Boolean operation does the circuit compute?

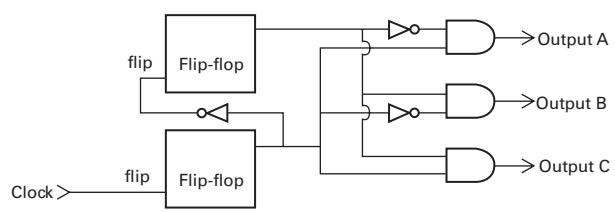


- *3. a.** If we were to purchase a flip-flop circuit from an electronic component store, we may find that it has an additional input called *flip*. When this input changes from a 0 to 1, the output flips state (if it was 0 it is now 1 and vice versa). However, when the flip input changes from 1 to a 0, nothing happens. Even though we may not know the details of the circuitry needed to accomplish this behavior, we could still

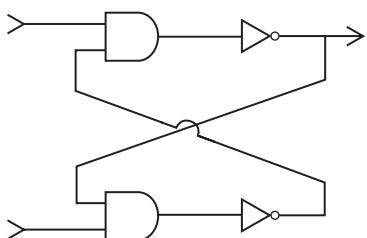
use this device as an abstract tool in other circuits. Consider the circuitry using two of the following flip-flops. If a pulse were sent on the circuit's input, the bottom flip-flop would change state. However, the second flip-flop would not change, since its input (received from the output of the NOT gate) went from a 1 to a 0. As a result, this circuit would now produce the outputs 0 and 1. A second pulse would flip the state of both flip-flops, producing an output of 1 and 0. What would be the output after a third pulse? After a fourth pulse?



- b.** It is often necessary to coordinate activities of various components within a computer. This is accomplished by connecting a pulsating signal (called a clock) to circuitry similar to part a. Additional gates (as shown) send signals in a coordinated fashion to other connected circuits. On studying this circuit, you should be able to confirm that on the 1st, 5th, 9th . . . pulses of the clock, a 1 will be sent on output A. On what pulses of the clock will a 1 be sent on output B? On what pulses of the clock will a 1 be sent on output C? On which output is a 1 sent on the 4th pulse of the clock?



4. Assume that both of the inputs in the following circuit are 1. Describe what would happen if the upper input were temporarily changed to 0. Describe what would happen if the lower input were temporarily changed to 0. Redraw the circuit using NAND gates.



5. The following table represents the addresses and contents (using hexadecimal notation) of some cells in a machine's main memory. Starting with this memory arrangement, follow the sequence of instructions and record the final contents of each of these memory cells:

| Address | Contents |
|---------|----------|
| 00 | AB |
| 01 | 53 |
| 02 | D6 |
| 03 | 02 |

- Step 1. Move the contents of the cell whose address is 03 to the cell at address 00.
 Step 2. Move the value 01 into the cell at address 02.
 Step 3. Move the value stored at address 01 into the cell at address 03.
6. How many cells can be in a computer's main memory if each cell's address can be represented by two hexadecimal digits? What if four hexadecimal digits are used?
7. What bit patterns are represented by the following hexadecimal notations?
 a. 8A9 b. DCB c. EF3
 d. A01 e. C99
8. What is the value of the least significant bit in the bit patterns represented by the following hexadecimal notations?
 a. 9A b. 90
 c. 1B d. 6E

9. Express the following bit patterns in hexadecimal notation:
 a. 10110100101101001011
 b. 000111100001
 c. 1111111011011011
10. Suppose a digital camera has a storage capacity of 500MB. How many black-and-white photographs could be stored in the camera if each consisted of 512 pixels per row and 512 pixels per column if each pixel required one bit of storage?
11. Suppose an image is represented on a display screen by a square array containing 256 columns and 256 rows of pixels. If for each pixel, 3 bytes are required to encode the color and 8 bits to encode the intensity, how many byte-size memory cells are required to hold the entire picture?
12. a. What are the advantages, if any, of using zoned-bit recording?
 b. What is the difference between seek time and access time?
13. Suppose that you want to create a backup of your entire data which is around 10GB. Would it be reasonable to use DVDs for the purpose of creating this backup? What about BDs (Blu-ray Disks)?
14. If each sector on a magnetic disk can store 512 bytes of data, how many sectors are required to store two pages of integers (perhaps 10 lines of 100 integers in each page) if every integer is represented by using four bytes?
15. How many bytes of storage space would be required to store a 20-page document containing details of employees, in which each page contains 100 records and every record is of 200 characters, if two byte Unicode characters were used?
16. In zoned-bit recording, why does the rate of data transfer vary, depending on the portion of the disk being used?
17. What is the average access time for a hard disk which has a rotation delay of 10 milliseconds and a seek time of 9 milliseconds?
18. Suppose a disk storage system consists of 5 platters with 10 tracks on each side and

- 8 sectors in each track. What is the capacity of the system? Assume every sector contains 512 bytes and data can be stored on both surfaces of each platter.
- 19.** Here is a message in ASCII. What does it say?
 01000011 01101111 01101101 01110000
 01110101 01110100 01100101 01110010
 00100000 01010011 01100011 01101001
 01100101 01101110 01100011 01100101
 00100001
- 20.** The following two messages are encoded in ASCII using one byte per character and then represented in hexadecimal notation. Are both the messages same?
 436F6D7075746572 436F6D7075736572
- 21.** Encode the following sentences in ASCII using one byte per character.
 a. Is 1 byte = 8 bits?
 b. Yes, a byte contains 8 bits!
- 22.** Combine the two sentences of the previous problem and express it in hexadecimal notation.
- 23.** List the hexadecimal representations of the integers from 20 to 27.
- 24.** a. Write the number 100 by representing 1 and 0 in ASCII.
 b. Write the number 255 in binary representation.
- 25.** What values have binary representations in which only one of the bits is 1? List the binary representations for the smallest six values with this property.
- *26.** Convert each of the following hexadecimal representations to binary representation and then to its equivalent base 10 representation:
 a. A b. 14 c. 1E
 d. 28 e. 32 f. 3C
 g. 46 h. 65 i. CA
 j. 12F k. 194 l. 1F9
- *27.** Convert each of the following base 10 representations to its equivalent binary representation:
 a. 110 b. 99 c. 72
 d. 81 e. 36
- *28.** Convert each of the following excess 32 representations to its equivalent base 10 representation:
 a. 011111 b. 100110 c. 111000
 d. 000101 e. 010101
- *29.** Convert each of the following base 10 representations to its equivalent excess sixteen representation:
 a. -12 b. 0 c. 10
 d. -8 e. 9
- *30.** Convert each of the following two's complement representations to its equivalent base 10 representation:
 a. 010101 b. 101010 c. 110110
 d. 011011 e. 111001
- *31.** Convert each of the following base 10 representations to its equivalent two's complement representation in which each value is represented in 8 bits:
 a. -27 b. 3 c. 21
 d. 8 e. -18
- *32.** Perform each of the following additions assuming the bit strings represent values in two's complement notation. Identify each case in which the answer is incorrect because of overflow.
 a. 00101 + 01000 b. 11111 + 00001
 c. 01111 + 00001 d. 10111 + 11010
 e. 11111 + 11111 f. 00111 + 01100
- *33.** Solve each of the following problems by translating the values into two's complement notation (using patterns of 5 bits), converting any subtraction problem to an equivalent addition problem, and performing that addition. Check your work by converting your answer to base 10 notation. (Watch out for overflow.)
 a. 5 + 1 b. 5 - 1 c. 12 - 5
 d. 8 - 7 e. 12 + 5 f. 5 - 11
- *34.** Convert each of the following binary representations into its equivalent base 10 representation:
 a. 11.11 b. 100.0101 c. 0.1101
 d. 1.0 e. 10.01
- *35.** Express each of the following values in binary notation:
 a. $5\frac{3}{4}$ b. $15\frac{15}{16}$ c. $5\frac{3}{8}$
 d. $1\frac{1}{4}$ e. $6\frac{5}{8}$
- *36.** Decode the following bit patterns using the floating-point format described in Figure 1.24:
 a. 01011001 b. 11001000
 c. 10101100 d. 00111001

- *37. Encode the following values using the 8-bit floating-point format described in Figure 1.24. Indicate each case in which a truncation error occurs.
- $-7\frac{1}{2}$
 - $\frac{1}{2}$
 - $-3\frac{3}{4}$
 - $\frac{7}{32}$
 - $\frac{31}{32}$
- *38. Assuming you are not restricted to using normalized form, list all the bit patterns that could be used to represent the value $3/8$ using the floating-point format described in Figure 1.24.
- *39. What is the best approximation to the square root of 2 that can be expressed in the 8-bit floating-point format described in Figure 1.24? What value is actually obtained if this approximation is squared by a machine using this floating-point format?
- *40. What is the best approximation to the value one-tenth that can be represented using the 8-bit floating-point format described in Figure 1.24?
- *41. Explain how errors can occur when measurements using the metric system are recorded in floating-point notation. For example, what if 110 cm was recorded in units of meters?
- *42. One of the bit patterns 01011 and 11011 represents a value stored in excess 16 notation and the other represents the same value stored in two's complement notation.
- What can be determined about this common value?
 - What is the relationship between a pattern representing a value stored in two's complement notation and the pattern representing the same value stored in excess notation when both systems use the same bit pattern length?
- *43. The three bit patterns 10000010, 01101000, and 00000010 are representations of the same value in two's complement, excess, and the 8-bit floating-point format presented in Figure 1.24, but not necessarily in that order. What is the common value, and which pattern is in which notation?
- *44. Which of the following values cannot be represented accurately in the floating-point format introduced in Figure 1.24?
- $6\frac{1}{2}$
 - $\frac{13}{16}$
 - 9
 - $\frac{17}{32}$
 - $\frac{15}{16}$
- *45. If you changed the length of the bit strings being used to represent integers in binary from 4 bits to 6 bits, what change would be made in the value of the largest integer you could represent? What if you were using two's complement notation?
- *46. What would be the hexadecimal representation of the largest memory address in a memory consisting of 4MB if each cell had a one-byte capacity?
- *47. What would be the encoded version of the message
 $xxy\ yyx\ xxy\ xxy\ yyx$
 if LZW compression, starting with the dictionary containing x, y, and a space (as described in Section 1.8), were used?
- *48. The following message was compressed using LZW compression with a dictionary whose first, second, and third entries are x, y, and space, respectively. What is the decompressed message?
 22123113431213536
- *49. If the message
 $xxy\ yyx\ xxy\ xxyy$
 were compressed using LZW with a starting dictionary whose first, second, and third entries were x, y, and space, respectively, what would be the entries in the final dictionary?
- *50. As we will learn in the next chapter, one means of transmitting bits over traditional telephone systems is to convert the bit patterns into sound, transfer the sound over the telephone lines, and then convert the sound back into bit patterns. Such techniques are limited to transfer rates of 57.6 Kbps. Is this sufficient for teleconferencing if the video is compressed using MPEG?
- *51. Encode the following sentences in ASCII using even parity by adding a parity bit at the high-order end of each character code:
- Does $100/5 = 20$?
 - The total cost is \$7.25.
- *52. The following message was originally transmitted with odd parity in each short bit string. In which strings have errors definitely occurred?
 11001 11011 10110 00000 11111 10001
 10101 00100 01110

- *53. Suppose a 24-bit code is generated by representing each symbol by three consecutive copies of its ASCII representation (for example, the symbol A is represented by the bit string 010000010100000101000001). What error-correcting properties does this new code have?
- *54. Using the error-correcting code described in Figure 1.28, decode the following words:
- a. 111010 110110
 - b. 101000 100110 001100
 - c. 011101 000110 000000 010100
 - d. 010010 001000 001110 101111
000000 110111 100110
 - e. 010011 000000 101001 100110
- *55. International currency exchange rates change frequently. Investigate current exchange rates, and update the currency converter script from Section 1.8 accordingly.
- *56. Find another currency not already included in the currency converter from Section 1.8. Acquire its current conversion rate and find its Unicode currency symbol on the web. Extend the script to convert this new currency.
- *57. If your web browser and text editor properly support Unicode and UTF-8, copy/paste the actual international currency symbols into the converter script of Section 1.8, in place of the cumbersome codes like, '\u00A3'. (If your software has trouble handling Unicode, you may get strange symbols in your text editor when you try to do this.)
- *58. The currency converter script of Section 1.8 uses the variable `dollars` to store the amount of money to be converted before performing each of the multiplications. This made the script one line longer than simply typing the integer quantity 1000 directly into each of the multiplication calculations. Why is it advantageous to create this extra variable ahead of time?
- *59. Write and test a Python script that given a number of bytes outputs the equivalent number of kilobytes, megabytes, gigabytes, and terabytes. Write and test a complementary script that given a number of terabytes outputs the equivalent number of GB, MB, KB, and bytes.
- *60. Write and test a Python script that given a number of minutes and seconds for a recording calculates the number of bits used to encode uncompressed, CD-quality stereo audio data of that length. (Review Section 1.4 for the necessary parameters and equations.)
- *61. Identify the error(s) in this Python script.
- ```
days_per_week = 7
weeks_per_year = 52
days_per_year = days_per_week **
 weeks_per_year
PRINT(days_per_year)
```

## Social Issues

The following questions are intended as a guide to the ethical/social/legal issues associated with the field of computing. The goal is not merely to answer these questions. You should also consider why you answered as you did and whether your justifications are consistent from one question to the next.

1. A truncation error has occurred in a critical situation, causing extensive damage and loss of life. Who is liable, if anyone? The designer of the hardware? The designer of the software? The programmer who actually wrote that part of the program? The person who decided to use the software in that particular application? What if the software had been corrected by the company that originally developed it, but that update had not been purchased and applied in the critical application? What if the software had been pirated?

2. Is it acceptable for an individual to ignore the possibility of truncation errors and their consequences when developing his or her own applications?
3. Was it ethical to develop software in the 1970s using only two digits to represent the year (such as using 76 to represent the year 1976), ignoring the fact that the software would be flawed as the turn of the century approached? Is it ethical today to use only three digits to represent the year (such as 982 for 1982 and 015 for 2015)? What about using only four digits?
4. Many argue that encoding information often dilutes or otherwise distorts the information, since it essentially forces the information to be quantified. They argue that a questionnaire in which subjects are required to record their opinions by responding within a scale from one to five is inherently flawed. To what extent is information quantifiable? Can the pros and cons of different locations for a waste disposal plant be quantified? Is the debate over nuclear power and nuclear waste quantifiable? Is it dangerous to base decisions on averages and other statistical analysis? Is it ethical for news agencies to report polling results without including the exact wording of the questions? Is it possible to quantify the value of a human life? Is it acceptable for a company to stop investing in the improvement of a product, even though additional investment could lower the possibility of a fatality relating to the product's use?
5. Should there be a distinction in the rights to collect and disseminate data depending on the form of the data? That is, should the right to collect and disseminate photographs, audio, or video be the same as the right to collect and disseminate text?
6. Whether intentional or not, a report submitted by a journalist usually reflects that journalist's bias. Often by changing only a few words, a story can be given either a positive or negative connotation. (Compare, "The majority of those surveyed opposed the referendum." to "A significant portion of those surveyed supported the referendum.") Is there a difference between altering a story (by leaving out certain points or carefully selecting words) and altering a photograph?
7. Suppose that the use of a data compression system results in the loss of subtle but significant items of information. What liability issues might be raised? How should they be resolved?

## Additional Reading

Drew, M., and Z. Li. *Fundamentals of Multimedia*. Upper Saddle River, NJ: Prentice-Hall, 2004.

Halsall, F. *Multimedia Communications*. Boston, MA: Addison-Wesley, 2001.

Hamacher, V. C., Z. G. Vranesic, and S. G. Zaky. *Computer Organization*, 5th ed. New York: McGraw-Hill, 2002.

Knuth, D. E. *The Art of Computer Programming*, Vol. 2, 3rd ed. Boston, MA: Addison-Wesley, 1998.

Long, B. *Complete Digital Photography*, 3rd ed. Hingham, MA: Charles River Media, 2005.

Miano, J. *Compressed Image File Formats*. New York: ACM Press, 1999.

Petzold, C. *CODE: The Hidden Language of Computer Hardware and Software*. Redmond, WA: Microsoft Press, 2000.

Salomon, D. *Data Compression: The Complete Reference*, 4th ed. New York: Springer, 2007.

Sayood, K. *Introduction to Data Compression*, 3rd ed. San Francisco, CA: Morgan Kaufmann, 2005.

# Discrete Mathematics

# Discrete Mathematics

Eighth Edition

**Richard Johnsonbaugh**

*DePaul University, Chicago*



Pearson

330 Hudson Street, NY, NY 10013

# Contents

## Preface XIII

## 1 Sets and Logic 1

---

- 1.1** Sets 2
  - 1.2** Propositions 14
  - 1.3** Conditional Propositions and Logical Equivalence 20
  - 1.4** Arguments and Rules of Inference 31
  - 1.5** Quantifiers 36
  - 1.6** Nested Quantifiers 49
- Problem-Solving Corner:** Quantifiers 57
- Chapter 1 Notes 58
  - Chapter 1 Review 58
  - Chapter 1 Self-Test 60
  - Chapter 1 Computer Exercises 60

## 2 Proofs 62

---

- 2.1** Mathematical Systems, Direct Proofs, and Counterexamples 63
  - 2.2** More Methods of Proof 72
- Problem-Solving Corner:** Proving Some Properties of Real Numbers 83
- 2.3** Resolution Proofs<sup>†</sup> 85
  - 2.4** Mathematical Induction 88
- Problem-Solving Corner:** Mathematical Induction 100
- 2.5** Strong Form of Induction and the Well-Ordering Property 102
- Chapter 2 Notes 109
  - Chapter 2 Review 109

---

<sup>†</sup>This section can be omitted without loss of continuity.

|                              |     |
|------------------------------|-----|
| Chapter 2 Self-Test          | 109 |
| Chapter 2 Computer Exercises | 110 |

## 3 Functions, Sequences, and Relations 111

---

|                                                          |                                   |     |
|----------------------------------------------------------|-----------------------------------|-----|
| <b>3.1</b>                                               | Functions                         | 111 |
| <b>Problem-Solving Corner:</b> Functions 128             |                                   |     |
| <b>3.2</b>                                               | Sequences and Strings             | 129 |
| <b>3.3</b>                                               | Relations                         | 141 |
| <b>3.4</b>                                               | Equivalence Relations             | 151 |
| <b>Problem-Solving Corner:</b> Equivalence Relations 158 |                                   |     |
| <b>3.5</b>                                               | Matrices of Relations             | 160 |
| <b>3.6</b>                                               | Relational Databases <sup>†</sup> | 165 |
|                                                          | Chapter 3 Notes                   | 170 |
|                                                          | Chapter 3 Review                  | 170 |
|                                                          | Chapter 3 Self-Test               | 171 |
|                                                          | Chapter 3 Computer Exercises      | 172 |

## 4 Algorithms 173

---

|                                                                           |                              |     |
|---------------------------------------------------------------------------|------------------------------|-----|
| <b>4.1</b>                                                                | Introduction                 | 173 |
| <b>4.2</b>                                                                | Examples of Algorithms       | 177 |
| <b>4.3</b>                                                                | Analysis of Algorithms       | 184 |
| <b>Problem-Solving Corner:</b> Design and Analysis<br>of an Algorithm 202 |                              |     |
| <b>4.4</b>                                                                | Recursive Algorithms         | 204 |
|                                                                           | Chapter 4 Notes              | 211 |
|                                                                           | Chapter 4 Review             | 211 |
|                                                                           | Chapter 4 Self-Test          | 212 |
|                                                                           | Chapter 4 Computer Exercises | 212 |

## 5 Introduction to Number Theory 214

---

|                                                   |                                                    |     |
|---------------------------------------------------|----------------------------------------------------|-----|
| <b>5.1</b>                                        | Divisors                                           | 214 |
| <b>5.2</b>                                        | Representations of Integers and Integer Algorithms | 224 |
| <b>5.3</b>                                        | The Euclidean Algorithm                            | 238 |
| <b>Problem-Solving Corner:</b> Making Postage 249 |                                                    |     |
| <b>5.4</b>                                        | The RSA Public-Key Cryptosystem                    | 250 |
|                                                   | Chapter 5 Notes                                    | 252 |
|                                                   | Chapter 5 Review                                   | 253 |
|                                                   | Chapter 5 Self-Test                                | 253 |
|                                                   | Chapter 5 Computer Exercises                       | 254 |

---

<sup>†</sup>This section can be omitted without loss of continuity.

**6****Counting Methods and the Pigeonhole Principle 255**

---

- 6.1** Basic Principles 255
- Problem-Solving Corner:** Counting 267
- 6.2** Permutations and Combinations 269
- Problem-Solving Corner:** Combinations 281
- 6.3** Generalized Permutations and Combinations 283
- 6.4** Algorithms for Generating Permutations and Combinations 289
- 6.5** Introduction to Discrete Probability<sup>†</sup> 297
- 6.6** Discrete Probability Theory<sup>†</sup> 301
- 6.7** Binomial Coefficients and Combinatorial Identities 313
- 6.8** The Pigeonhole Principle 319
- Chapter 6 Notes 324
- Chapter 6 Review 324
- Chapter 6 Self-Test 325
- Chapter 6 Computer Exercises 326

**7****Recurrence Relations 327**

---

- 7.1** Introduction 327
- 7.2** Solving Recurrence Relations 338
- Problem-Solving Corner:** Recurrence Relations 350
- 7.3** Applications to the Analysis of Algorithms 353
- 7.4** The Closest-Pair Problem<sup>†</sup> 365
- Chapter 7 Notes 370
- Chapter 7 Review 371
- Chapter 7 Self-Test 371
- Chapter 7 Computer Exercises 372

**8****Graph Theory 373**

---

- 8.1** Introduction 373
- 8.2** Paths and Cycles 384
- Problem-Solving Corner:** Graphs 395
- 8.3** Hamiltonian Cycles and the Traveling Salesperson Problem 396
- 8.4** A Shortest-Path Algorithm 405
- 8.5** Representations of Graphs 410
- 8.6** Isomorphisms of Graphs 415
- 8.7** Planar Graphs 422
- 8.8** Instant Insanity<sup>†</sup> 429

---

<sup>†</sup>This section can be omitted without loss of continuity.

|                              |     |
|------------------------------|-----|
| Chapter 8 Notes              | 433 |
| Chapter 8 Review             | 434 |
| Chapter 8 Self-Test          | 435 |
| Chapter 8 Computer Exercises | 436 |

**9****Trees 438**

|                                          |                                                 |     |
|------------------------------------------|-------------------------------------------------|-----|
| <b>9.1</b>                               | Introduction                                    | 438 |
| <b>9.2</b>                               | Terminology and Characterizations of Trees      | 445 |
| <b>Problem-Solving Corner:</b> Trees 450 |                                                 |     |
| <b>9.3</b>                               | Spanning Trees                                  | 452 |
| <b>9.4</b>                               | Minimal Spanning Trees                          | 459 |
| <b>9.5</b>                               | Binary Trees                                    | 465 |
| <b>9.6</b>                               | Tree Traversals                                 | 471 |
| <b>9.7</b>                               | Decision Trees and the Minimum Time for Sorting | 477 |
| <b>9.8</b>                               | Isomorphisms of Trees                           | 483 |
| <b>9.9</b>                               | Game Trees <sup>†</sup>                         | 493 |
|                                          | Chapter 9 Notes                                 | 502 |
|                                          | Chapter 9 Review                                | 502 |
|                                          | Chapter 9 Self-Test                             | 503 |
|                                          | Chapter 9 Computer Exercises                    | 505 |

**10****Network Models 506**

|                                             |                               |     |
|---------------------------------------------|-------------------------------|-----|
| <b>10.1</b>                                 | Introduction                  | 506 |
| <b>10.2</b>                                 | A Maximal Flow Algorithm      | 511 |
| <b>10.3</b>                                 | The Max Flow, Min Cut Theorem | 519 |
| <b>10.4</b>                                 | Matching                      | 523 |
| <b>Problem-Solving Corner:</b> Matching 528 |                               |     |
|                                             | Chapter 10 Notes              | 529 |
|                                             | Chapter 10 Review             | 530 |
|                                             | Chapter 10 Self-Test          | 530 |
|                                             | Chapter 10 Computer Exercises | 531 |

**11****Boolean Algebras and Combinatorial Circuits 532**

|                                                     |                                             |     |
|-----------------------------------------------------|---------------------------------------------|-----|
| <b>11.1</b>                                         | Combinatorial Circuits                      | 532 |
| <b>11.2</b>                                         | Properties of Combinatorial Circuits        | 539 |
| <b>11.3</b>                                         | Boolean Algebras                            | 544 |
| <b>Problem-Solving Corner:</b> Boolean Algebras 549 |                                             |     |
| <b>11.4</b>                                         | Boolean Functions and Synthesis of Circuits | 551 |
| <b>11.5</b>                                         | Applications                                | 556 |

---

<sup>†</sup>This section can be omitted without loss of continuity.

|                               |     |
|-------------------------------|-----|
| Chapter 11 Notes              | 564 |
| Chapter 11 Review             | 565 |
| Chapter 11 Self-Test          | 565 |
| Chapter 11 Computer Exercises | 567 |

# 12

## Automata, Grammars, and Languages 568

---

|             |                                               |     |
|-------------|-----------------------------------------------|-----|
| <b>12.1</b> | Sequential Circuits and Finite-State Machines | 568 |
| <b>12.2</b> | Finite-State Automata                         | 574 |
| <b>12.3</b> | Languages and Grammars                        | 579 |
| <b>12.4</b> | Nondeterministic Finite-State Automata        | 589 |
| <b>12.5</b> | Relationships Between Languages and Automata  | 595 |
|             | Chapter 12 Notes                              | 601 |
|             | Chapter 12 Review                             | 602 |
|             | Chapter 12 Self-Test                          | 602 |
|             | Chapter 12 Computer Exercises                 | 603 |

## Appendix 605

### A

#### Matrices 605

---

### B

#### Algebra Review 609

---

### C

#### Pseudocode 620

---

## References 627

## Hints and Solutions to Selected Exercises 633

## Index 735

*This page intentionally left blank*

*Dedication*

To Pat, my wife, for her continuous support through my many book projects, for formally and informally copy-editing my books, for maintaining good cheer throughout, and for preventing all *egregious* mistakes that would have otherwise found their way into print. Her contributions are deeply appreciated.

*This page intentionally left blank*

*This page intentionally left blank*



# Chapter 1

## SETS AND LOGIC

- 1.1** Sets
- 1.2** Propositions
- 1.3** Conditional Propositions and Logical Equivalence
- 1.4** Arguments and Rules of Inference
- 1.5** Quantifiers
- 1.6** Nested Quantifiers

### Go Online

For more on logic, see  
[goo.gl/F7b35e](http://goo.gl/F7b35e)

Chapter 1 begins with sets. A **set** is a collection of objects; order is not taken into account. Discrete mathematics is concerned with objects such as graphs (sets of vertices and edges) and Boolean algebras (sets with certain operations defined on them). In this chapter, we introduce set terminology and notation. In Chapter 2, we treat sets more formally after discussing proof and proof techniques. However, in Section 1.1, we provide a taste of the logic and proofs to come in the remainder of Chapter 1 and in Chapter 2.

**Logic** is the study of reasoning; it is specifically concerned with whether reasoning is correct. Logic focuses on the relationship among statements as opposed to the content of any particular statement. Consider, for example, the following argument:

All mathematicians wear sandals.

Anyone who wears sandals is an algebraist.

Therefore, all mathematicians are algebraists.

Technically, logic is of no help in determining whether any of these statements is true; however, if the first two statements are true, logic assures us that the statement,

All mathematicians are algebraists,

is also true.

Logic is essential in reading and developing proofs, which we explore in detail in Chapter 2. An understanding of logic can also be useful in clarifying ordinary writing. For example, at one time, the following ordinance was in effect in Naperville, Illinois: “It shall be unlawful for any person to keep more than three dogs and three cats upon his property within the city.” Was one of the citizens, who owned five dogs and no cats, in violation of the ordinance? Think about this question now; then analyze it (see Exercise 75, Section 1.2) after reading Section 1.2.

## 1.1 Sets

### Go Online

For more on sets, see  
[goo.gl/F7b35e](http://goo.gl/F7b35e)

The concept of set is basic to all of mathematics and mathematical applications. A **set** is simply a collection of objects. The objects are sometimes referred to as elements or members. If a set is finite and not too large, we can describe it by listing the elements in it. For example, the equation

$$A = \{1, 2, 3, 4\} \quad (1.1.1)$$

describes a set  $A$  made up of the four elements 1, 2, 3, and 4. A set is determined by its elements and not by any particular order in which the elements might be listed. Thus the set  $A$  might just as well be specified as  $A = \{1, 3, 4, 2\}$ . The elements making up a set are assumed to be distinct, and although for some reason we may have duplicates in our list, only one occurrence of each element is in the set. For this reason we may also describe the set  $A$  defined in (1.1.1) as  $A = \{1, 2, 2, 3, 4\}$ .

If a set is a large finite set or an infinite set, we can describe it by listing a property necessary for membership. For example, the equation

$$B = \{x \mid x \text{ is a positive, even integer}\} \quad (1.1.2)$$

describes the set  $B$  made up of all positive, even integers; that is,  $B$  consists of the integers 2, 4, 6, and so on. The vertical bar “ $\mid$ ” is read “such that.” Equation (1.1.2) would be read “ $B$  equals the set of all  $x$  such that  $x$  is a positive, even integer.” Here the property necessary for membership is “is a positive, even integer.” Note that the property appears after the vertical bar. The notation in (1.1.2) is called **set-builder notation**.

A set may contain *any* kind of elements whatsoever, and they need *not* be of the same “type.” For example,

$$\{4.5, \text{Lady Gaga}, \pi, 14\}$$

is a perfectly fine set. It consists of four elements: the number 4.5, the person Lady Gaga, the number  $\pi (= 3.1415 \dots)$ , and the number 14.

A set may contain elements that are themselves sets. For example, the set

$$\{3, \{5, 1\}, 12, \{\pi, 4.5, 40, 16\}, \text{Henry Cavill}\}$$

consists of five elements: the number 3, the set  $\{5, 1\}$ , the number 12, the set  $\{\pi, 4.5, 40, 16\}$ , and the person Henry Cavill.

Some sets of numbers that occur frequently in mathematics generally, and in discrete mathematics in particular, are shown in Figure 1.1.1. The symbol **Z** comes from the German word, *Zahlen*, for *integer*. Rational numbers are quotients of integers, thus **Q** for *quotient*. The set of real numbers **R** can be depicted as consisting of all points on a straight line extending indefinitely in either direction (see Figure 1.1.2).†

| Symbol   | Set              | Example of Members                                 |
|----------|------------------|----------------------------------------------------|
| <b>Z</b> | Integers         | -3, 0, 2, 145                                      |
| <b>Q</b> | Rational numbers | -1/3, 0, 24/15                                     |
| <b>R</b> | Real numbers     | -3, -1.766, 0, 4/15, $\sqrt{2}$ , 2.666 ..., $\pi$ |

**Figure 1.1.1** Sets of numbers.

†The real numbers can be constructed by starting with a more primitive notion such as “set” or “integer,” or they can be obtained by stating properties (axioms) they are assumed to obey. For our purposes, it suffices to think of the real numbers as points on a straight line. The construction of the real numbers and the axioms for the real numbers are beyond the scope of this book.

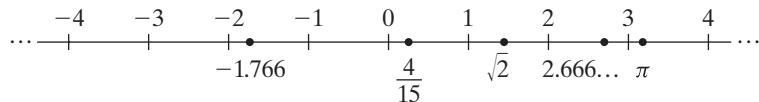


Figure 1.1.2 The real number line.

To denote the negative numbers that belong to one of  $\mathbf{Z}$ ,  $\mathbf{Q}$ , or  $\mathbf{R}$ , we use the superscript minus. For example,  $\mathbf{Z}^-$  denotes the set of negative integers, namely  $-1, -2, -3, \dots$ . Similarly, to denote the positive numbers that belong to one of the three sets, we use the superscript plus. For example,  $\mathbf{Q}^+$  denotes the set of positive rational numbers. To denote the nonnegative numbers that belong to one of the three sets, we use the superscript *nonneg*. For example,  $\mathbf{Z}^{\text{nonneg}}$  denotes the set of nonnegative integers, namely  $0, 1, 2, 3, \dots$ .

If  $X$  is a finite set, we let  $|X| =$  number of elements in  $X$ . We call  $|X|$  the **cardinality** of  $X$ . There is also a notion of cardinality of infinite sets, although we will not discuss it in this book. For example, the cardinality of the integers,  $\mathbf{Z}$ , is denoted  $\aleph_0$ , read “aleph null.” Aleph is the first letter of the Hebrew alphabet.

**Example 1.1.1**

For the set  $A$  in (1.1.1), we have  $|A| = 4$ , and the cardinality of  $A$  is 4. The cardinality of the set  $\{\mathbf{R}, \mathbf{Z}\}$  is 2 since it contains two elements, namely the two sets  $\mathbf{R}$  and  $\mathbf{Z}$ . 

Given a description of a set  $X$  such as (1.1.1) or (1.1.2) and an element  $x$ , we can determine whether or not  $x$  belongs to  $X$ . If the members of  $X$  are listed as in (1.1.1), we simply look to see whether or not  $x$  appears in the listing. In a description such as (1.1.2), we check to see whether the element  $x$  has the property listed. If  $x$  is in the set  $X$ , we write  $x \in X$ , and if  $x$  is not in  $X$ , we write  $x \notin X$ . For example,  $3 \in \{1, 2, 3, 4\}$ , but  $3 \notin \{x \mid x \text{ is a positive, even integer}\}$ .

The set with no elements is called the **empty** (or **null** or **void**) **set** and is denoted  $\emptyset$ . Thus  $\emptyset = \{ \}$ .

Two sets  $X$  and  $Y$  are **equal** and we write  $X = Y$  if  $X$  and  $Y$  have the same elements. To put it another way,  $X = Y$  if the following two conditions hold:

- For every  $x$ , if  $x \in X$ , then  $x \in Y$ ,

and

- For every  $x$ , if  $x \in Y$ , then  $x \in X$ .

The first condition ensures that every element of  $X$  is an element of  $Y$ , and the second condition ensures that every element of  $Y$  is an element of  $X$ .

**Example 1.1.2**

If  $A = \{1, 3, 2\}$  and  $B = \{2, 3, 2, 1\}$ , by inspection,  $A$  and  $B$  have the same elements. Therefore  $A = B$ . 

**Example 1.1.3**

Show that if  $A = \{x \mid x^2 + x - 6 = 0\}$  and  $B = \{2, -3\}$ , then  $A = B$ .

**SOLUTION** According to the criteria in the paragraph immediately preceding Example 1.1.2, we must show that for every  $x$ ,

$$\text{if } x \in A, \text{ then } x \in B, \quad (1.1.3)$$

and for every  $x$ ,

$$\text{if } x \in B, \text{ then } x \in A. \quad (1.1.4)$$

To verify condition (1.1.3), suppose that  $x \in A$ . Then

$$x^2 + x - 6 = 0.$$

Solving for  $x$ , we find that  $x = 2$  or  $x = -3$ . In either case,  $x \in B$ . Therefore, condition (1.1.3) holds.

To verify condition (1.1.4), suppose that  $x \in B$ . Then  $x = 2$  or  $x = -3$ . If  $x = 2$ , then

$$x^2 + x - 6 = 2^2 + 2 - 6 = 0.$$

Therefore,  $x \in A$ . If  $x = -3$ , then

$$x^2 + x - 6 = (-3)^2 + (-3) - 6 = 0.$$

Again,  $x \in A$ . Therefore, condition (1.1.4) holds. We conclude that  $A = B$ . 

For a set  $X$  to *not* be equal to a set  $Y$  (written  $X \neq Y$ ),  $X$  and  $Y$  must *not* have the same elements: There must be at least one element in  $X$  that is not in  $Y$  or at least one element in  $Y$  that is not in  $X$  (or both).

#### **Example 1.1.4**

Let  $A = \{1, 2, 3\}$  and  $B = \{2, 4\}$ . Then  $A \neq B$  since there is at least one element in  $A$  (1 for example) that is not in  $B$ . [Another way to see that  $A \neq B$  is to note that there is at least one element in  $B$  (namely 4) that is not in  $A$ .] 

Suppose that  $X$  and  $Y$  are sets. If every element of  $X$  is an element of  $Y$ , we say that  $X$  is a **subset** of  $Y$  and write  $X \subseteq Y$ . In other words,  $X$  is a subset of  $Y$  if for every  $x$ , if  $x \in X$ , then  $x \in Y$ .

#### **Example 1.1.5**

If  $C = \{1, 3\}$  and  $A = \{1, 2, 3, 4\}$ , by inspection, every element of  $C$  is an element of  $A$ . Therefore,  $C$  is a subset of  $A$  and we write  $C \subseteq A$ . 

#### **Example 1.1.6**

Let  $X = \{x \mid x^2 + x - 2 = 0\}$ . Show that  $X \subseteq \mathbf{Z}$ .

**SOLUTION** We must show that for every  $x$ , if  $x \in X$ , then  $x \in \mathbf{Z}$ . If  $x \in X$ , then  $x^2 + x - 2 = 0$ . Solving for  $x$ , we obtain  $x = 1$  or  $x = -2$ . In either case,  $x \in \mathbf{Z}$ . Therefore, for every  $x$ , if  $x \in X$ , then  $x \in \mathbf{Z}$ . We conclude that  $X$  is a subset of  $\mathbf{Z}$  and we write  $X \subseteq \mathbf{Z}$ . 

#### **Example 1.1.7**

The set of integers  $\mathbf{Z}$  is a subset of the set of rational numbers  $\mathbf{Q}$ . If  $n \in \mathbf{Z}$ ,  $n$  can be expressed as a quotient of integers, for example,  $n = n/1$ . Therefore  $n \in \mathbf{Q}$  and  $\mathbf{Z} \subseteq \mathbf{Q}$ . 

#### **Example 1.1.8**

The set of rational numbers  $\mathbf{Q}$  is a subset of the set of real numbers  $\mathbf{R}$ . If  $x \in \mathbf{Q}$ ,  $x$  corresponds to a point on the number line (see Figure 1.1.2) so  $x \in \mathbf{R}$ . 

For  $X$  to *not* be a subset of  $Y$ , there must be at least one member of  $X$  that is not in  $Y$ .

#### **Example 1.1.9**

Let  $X = \{x \mid 3x^2 - x - 2 = 0\}$ . Show that  $X$  is not a subset of  $\mathbf{Z}$ .

**SOLUTION** If  $x \in X$ , then  $3x^2 - x - 2 = 0$ . Solving for  $x$ , we obtain  $x = 1$  or  $x = -2/3$ . Taking  $x = -2/3$ , we have  $x \in X$  but  $x \notin \mathbf{Z}$ . Therefore,  $X$  is not a subset of  $\mathbf{Z}$ . 

Any set  $X$  is a subset of itself, since any element in  $X$  is in  $X$ . Also, the empty set is a subset of every set. If  $\emptyset$  is *not* a subset of some set  $Y$ , according to the discussion preceding Example 1.1.9, there would have to be at least one member of  $\emptyset$  that is not in  $Y$ . But this cannot happen because the empty set, by definition, has no members.

Notice the difference between the terms “subset” and “element of.” The set  $X$  is a *subset* of the set  $Y$  ( $X \subseteq Y$ ), if every element of  $X$  is an element of  $Y$ ;  $x$  is an *element of*  $X$  ( $x \in X$ ), if  $x$  is a member of the set  $X$ .

#### Example 1.1.10

Let  $X = \{1, 3, 5, 7\}$  and  $Y = \{1, 2, 3, 4, 5, 6, 7\}$ . Then  $X \subseteq Y$  since every element of  $X$  is an element of  $Y$ . But  $X \notin Y$ , since the *set*  $X$  is not a member of  $Y$ . Also,  $1 \in X$ , but  $1$  is not a subset of  $X$ . Notice the difference between the number  $1$  and the *set*  $\{1\}$ . The set  $\{1\}$  is a subset of  $X$ . 

If  $X$  is a subset of  $Y$  and  $X$  does not equal  $Y$ , we say that  $X$  is a **proper subset** of  $Y$  and write  $X \subset Y$ .

#### Example 1.1.11

Let  $C = \{1, 3\}$  and  $A = \{1, 2, 3, 4\}$ . Then  $C$  is a proper subset of  $A$  since  $C$  is a subset of  $A$  but  $C$  does not equal  $A$ . We write  $C \subset A$ . 

#### Example 1.1.12

Example 1.1.7 showed that  $\mathbf{Z}$  is a subset of  $\mathbf{Q}$ . In fact,  $\mathbf{Z}$  is a proper subset of  $\mathbf{Q}$  because, for example,  $1/2 \in \mathbf{Q}$ , but  $1/2 \notin \mathbf{Z}$ . 

#### Example 1.1.13

Example 1.1.8 showed that  $\mathbf{Q}$  is a subset of  $\mathbf{R}$ . In fact,  $\mathbf{Q}$  is a proper subset of  $\mathbf{R}$  because, for example,  $\sqrt{2} \in \mathbf{R}$ , but  $\sqrt{2} \notin \mathbf{Q}$ . (In Example 2.2.3, we will show that  $\sqrt{2}$  is not the quotient of integers). 

The set of all subsets (proper or not) of a set  $X$ , denoted  $\mathcal{P}(X)$ , is called the **power set** of  $X$ .

#### Example 1.1.14

If  $A = \{a, b, c\}$ , the members of  $\mathcal{P}(A)$  are

$$\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}.$$

All but  $\{a, b, c\}$  are proper subsets of  $A$ . 

In Example 1.1.14,  $|A| = 3$  and  $|\mathcal{P}(A)| = 2^3 = 8$ . In Section 2.4 (Theorem 2.4.6), we will give a formal proof that this result holds in general; that is, the power set of a set with  $n$  elements has  $2^n$  elements.

Given two sets  $X$  and  $Y$ , there are various set operations involving  $X$  and  $Y$  that can produce a new set. The set

$$X \cup Y = \{x \mid x \in X \text{ or } x \in Y\}$$

is called the **union** of  $X$  and  $Y$ . The union consists of all elements belonging to either  $X$  or  $Y$  (or both).

The set

$$X \cap Y = \{x \mid x \in X \text{ and } x \in Y\}$$

is called the **intersection** of  $X$  and  $Y$ . The intersection consists of all elements belonging to both  $X$  and  $Y$ .

The set

$$X - Y = \{x \mid x \in X \text{ and } x \notin Y\}$$

is called the **difference** (or **relative complement**). The difference  $X - Y$  consists of all elements in  $X$  that are not in  $Y$ .

### Example 1.1.15

If  $A = \{1, 3, 5\}$  and  $B = \{4, 5, 6\}$ , then

$$A \cup B = \{1, 3, 4, 5, 6\}$$

$$A \cap B = \{5\}$$

$$A - B = \{1, 3\}$$

$$B - A = \{4, 6\}.$$

Notice that, in general,  $A - B \neq B - A$ .

### Example 1.1.16

Since  $\mathbf{Q} \subseteq \mathbf{R}$ ,

$$\mathbf{R} \cup \mathbf{Q} = \mathbf{R}$$

$$\mathbf{R} \cap \mathbf{Q} = \mathbf{Q}$$

$$\mathbf{Q} - \mathbf{R} = \emptyset.$$

The set  $\mathbf{R} - \mathbf{Q}$ , called the set of **irrational numbers**, consists of all real numbers that are not rational.

We call a set  $\mathcal{S}$ , whose elements are sets, a **collection of sets** or a **family of sets**. For example, if

$$\mathcal{S} = \{\{1, 2\}, \{1, 3\}, \{1, 7, 10\}\},$$

then  $\mathcal{S}$  is a collection or family of sets. The set  $\mathcal{S}$  consists of the sets

$$\{1, 2\}, \{1, 3\}, \{1, 7, 10\}.$$

Sets  $X$  and  $Y$  are **disjoint** if  $X \cap Y = \emptyset$ . A collection of sets  $\mathcal{S}$  is said to be **pairwise disjoint** if, whenever  $X$  and  $Y$  are distinct sets in  $\mathcal{S}$ ,  $X$  and  $Y$  are disjoint.

### Example 1.1.17

The sets  $\{1, 4, 5\}$  and  $\{2, 6\}$  are disjoint. The collection of sets  $\mathcal{S} = \{\{1, 4, 5\}, \{2, 6\}, \{3\}, \{7, 8\}\}$  is pairwise disjoint.

Sometimes we are dealing with sets, all of which are subsets of a set  $U$ . This set  $U$  is called a **universal set** or a **universe**. The set  $U$  must be explicitly given or inferred from the context. Given a universal set  $U$  and a subset  $X$  of  $U$ , the set  $U - X$  is called the **complement** of  $X$  and is written  $\bar{X}$ .

### Example 1.1.18

Let  $A = \{1, 3, 5\}$ . If  $U$ , a universal set, is specified as  $U = \{1, 2, 3, 4, 5\}$ , then  $\bar{A} = \{2, 4\}$ . If, on the other hand, a universal set is specified as  $U = \{1, 3, 5, 7, 9\}$ , then  $\bar{A} = \{7, 9\}$ . The complement obviously depends on the universe in which we are working.

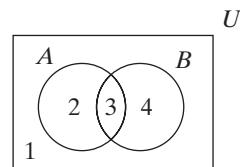
### Example 1.1.19

Let the universal set be  $\mathbf{Z}$ . Then  $\overline{\mathbf{Z}^-}$ , the complement of the set of negative integers, is  $\mathbf{Z}^{nonneg}$ , the set of nonnegative integers.

**Go Online**

For more on Venn diagrams, see  
[goo.gl/F7b35e](http://goo.gl/F7b35e)

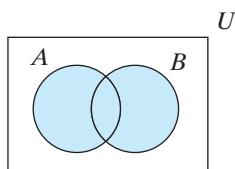
**Venn diagrams** provide pictorial views of sets. In a Venn diagram, a rectangle depicts a universal set (see Figure 1.1.3). Subsets of the universal set are drawn as circles. The inside of a circle represents the members of that set. In Figure 1.1.3 we see two sets  $A$  and  $B$  within the universal set  $U$ . Region 1 represents  $(A \cup B)^c$ , the elements in neither  $A$  nor  $B$ . Region 2 represents  $A - B$ , the elements in  $A$  but not in  $B$ . Region 3 represents  $A \cap B$ , the elements in both  $A$  and  $B$ . Region 4 represents  $B - A$ , the elements in  $B$  but not in  $A$ .



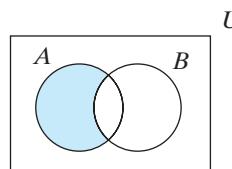
**Figure 1.1.3** A Venn diagram.

**Example 1.1.20**

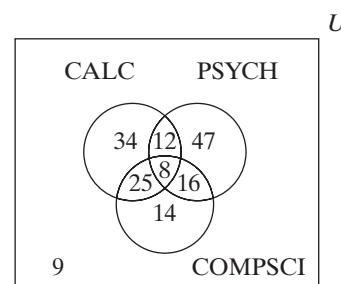
Particular regions in Venn diagrams are depicted by shading. The set  $A \cup B$  is shown in Figure 1.1.4, and Figure 1.1.5 represents the set  $A - B$ .



**Figure 1.1.4** A Venn diagram of  $A \cup B$ .



**Figure 1.1.5** A Venn diagram of  $A - B$ .



**Figure 1.1.6** A Venn diagram of three sets CALC, PSYCH, and COMPSCI. The numbers show how many students belong to the particular region depicted.

To represent three sets, we use three overlapping circles (see Figure 1.1.6).

**Example 1.1.21**

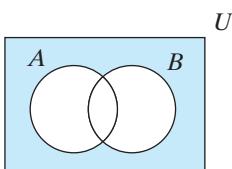
Among a group of 165 students, 8 are taking calculus, psychology, and computer science; 33 are taking calculus and computer science; 20 are taking calculus and psychology; 24 are taking psychology and computer science; 79 are taking calculus; 83 are taking psychology; and 63 are taking computer science. How many are taking none of the three subjects?

**SOLUTION** Let CALC, PSYCH, and COMPSCI denote the sets of students taking calculus, psychology, and computer science, respectively. Let  $U$  denote the set of all 165 students (see Figure 1.1.6). Since 8 students are taking calculus, psychology, and computer science, we write 8 in the region representing  $\text{CALC} \cap \text{PSYCH} \cap \text{COMPSCI}$ . Of the 33 students taking calculus and computer science, 8 are also taking psychology; thus 25 are taking calculus and computer science but not psychology. We write 25 in the region representing  $\text{CALC} \cap \overline{\text{PSYCH}} \cap \text{COMPSCI}$ . Similarly, we write 12 in the region representing  $\text{CALC} \cap \text{PSYCH} \cap \overline{\text{COMPSCI}}$  and 16 in the region representing  $\overline{\text{CALC}} \cap \text{PSYCH} \cap \text{COMPSCI}$ . Of the 79 students taking calculus, 45 have now been accounted for. This leaves 34 students taking only calculus. We write 34 in the region representing  $\text{CALC} \cap \overline{\text{PSYCH}} \cap \overline{\text{COMPSCI}}$ . Similarly, we write 47 in the region representing  $\overline{\text{CALC}} \cap \text{PSYCH} \cap \overline{\text{COMPSCI}}$  and 14 in the region representing

$\overline{\text{CALC}} \cap \overline{\text{PSYCH}} \cap \text{COMPSCI}$ . At this point, 156 students have been accounted for. This leaves 9 students taking none of the three subjects. ▲

Venn diagrams can also be used to visualize certain properties of sets. For example, by sketching both  $(A \cup B)$  and  $\overline{A} \cap \overline{B}$  (see Figure 1.1.7), we see that these sets are equal. A formal proof would show that for every  $x$ , if  $x \in (A \cup B)$ , then  $x \in \overline{A} \cap \overline{B}$ , and if  $x \in \overline{A} \cap \overline{B}$ , then  $x \in (A \cup B)$ . We state many useful properties of sets as Theorem 1.1.22.

### Theorem 1.1.22



**Figure 1.1.7** The shaded region depicts both  $(A \cup B)$  and  $\overline{A} \cap \overline{B}$ ; thus these sets are equal.

Let  $U$  be a universal set and let  $A$ ,  $B$ , and  $C$  be subsets of  $U$ . The following properties hold.

(a) Associative laws:

$$(A \cup B) \cup C = A \cup (B \cup C), \quad (A \cap B) \cap C = A \cap (B \cap C)$$

(b) Commutative laws:

$$A \cup B = B \cup A, \quad A \cap B = B \cap A$$

(c) Distributive laws:

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C), \quad A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

(d) Identity laws:

$$A \cup \emptyset = A, \quad A \cap U = A$$

(e) Complement laws:

$$A \cup \overline{A} = U, \quad A \cap \overline{A} = \emptyset$$

(f) Idempotent laws:

$$A \cup A = A, \quad A \cap A = A$$

(g) Bound laws:

$$A \cup U = U, \quad A \cap \emptyset = \emptyset$$

(h) Absorption laws:

$$A \cup (A \cap B) = A, \quad A \cap (A \cup B) = A$$

(i) Involution law:

$$\overline{\overline{A}} = A^\dagger$$

(j) 0/1 laws:

$$\overline{\emptyset} = U, \quad \overline{U} = \emptyset$$

(k) De Morgan's laws for sets:

$$\overline{(A \cup B)} = \overline{A} \cap \overline{B}, \quad \overline{(A \cap B)} = \overline{A} \cup \overline{B}$$

### Go Online

For a biography of De Morgan, see [goo.gl/F7b35e](http://goo.gl/F7b35e)

**Proof** The proofs are left as exercises (Exercises 46–56, Section 2.1) to be done after more discussion of logic and proof techniques. ▲

We define the union of a collection of sets  $\mathcal{S}$  to be those elements  $x$  belonging to at least one set  $X$  in  $\mathcal{S}$ . Formally,

$$\cup \mathcal{S} = \{x \mid x \in X \text{ for some } X \in \mathcal{S}\}.$$

---

<sup>†</sup> $\overline{\overline{A}}$  denotes the complement of the complement of  $A$ , that is,  $\overline{\overline{A}} = \overline{(A)}$ .

Similarly, we define the intersection of a collection of sets  $\mathcal{S}$  to be those elements  $x$  belonging to every set  $X$  in  $\mathcal{S}$ . Formally,

$$\cap \mathcal{S} = \{x \mid x \in X \text{ for all } X \in \mathcal{S}\}.$$

**Example 1.1.23**

Let  $\mathcal{S} = \{\{1, 2\}, \{1, 3\}, \{1, 7, 10\}\}$ . Then  $\cup \mathcal{S} = \{1, 2, 3, 7, 10\}$  since each of the elements 1, 2, 3, 7, 10 belongs to at least one set in  $\mathcal{S}$ , and no other element belongs to any of the sets in  $\mathcal{S}$ . Also  $\cap \mathcal{S} = \{1\}$  since only the element 1 belongs to every set in  $\mathcal{S}$ .  $\blacktriangleleft$

If

$$\mathcal{S} = \{A_1, A_2, \dots, A_n\},$$

we write

$$\cup \mathcal{S} = \bigcup_{i=1}^n A_i, \quad \cap \mathcal{S} = \bigcap_{i=1}^n A_i,$$

and if

$$\mathcal{S} = \{A_1, A_2, \dots\},$$

we write

$$\cup \mathcal{S} = \bigcup_{i=1}^{\infty} A_i, \quad \cap \mathcal{S} = \bigcap_{i=1}^{\infty} A_i.$$

**Example 1.1.24**

For  $i \geq 1$ , define  $A_i = \{i, i + 1, \dots\}$  and  $\mathcal{S} = \{A_1, A_2, \dots\}$ . As examples,  $A_1 = \{1, 2, 3, \dots\}$  and  $A_2 = \{2, 3, 4, \dots\}$ . Then

$$\cup \mathcal{S} = \bigcup_{i=1}^{\infty} A_i = \{1, 2, \dots\}, \quad \cap \mathcal{S} = \bigcap_{i=1}^{\infty} A_i = \emptyset. \quad \blacktriangleleft$$

A partition of a set  $X$  divides  $X$  into nonoverlapping subsets. More formally, a collection  $\mathcal{S}$  of nonempty subsets of  $X$  is said to be a **partition** of the set  $X$  if every element in  $X$  belongs to exactly one member of  $\mathcal{S}$ . Notice that if  $\mathcal{S}$  is a partition of  $X$ ,  $\mathcal{S}$  is pairwise disjoint and  $\cup \mathcal{S} = X$ .

**Example 1.1.25**

Since each element of  $X = \{1, 2, 3, 4, 5, 6, 7, 8\}$  is in exactly one member of  $\mathcal{S} = \{\{1, 4, 5\}, \{2, 6\}, \{3\}, \{7, 8\}\}$ ,  $\mathcal{S}$  is a partition of  $X$ .  $\blacktriangleleft$

At the beginning of this section, we pointed out that a set is an unordered collection of elements; that is, a set is determined by its elements and not by any particular order in which the elements are listed. Sometimes, however, we do want to take order into account. An **ordered pair** of elements, written  $(a, b)$ , is considered distinct from the ordered pair  $(b, a)$ , unless, of course,  $a = b$ . To put it another way,  $(a, b) = (c, d)$  precisely when  $a = c$  and  $b = d$ . If  $X$  and  $Y$  are sets, we let  $X \times Y$  denote the set of all ordered pairs  $(x, y)$  where  $x \in X$  and  $y \in Y$ . We call  $X \times Y$  the **Cartesian product** of  $X$  and  $Y$ .

**Example 1.1.26**

If  $X = \{1, 2, 3\}$  and  $Y = \{a, b\}$ , then

$$X \times Y = \{(1, a), (1, b), (2, a), (2, b), (3, a), (3, b)\}$$

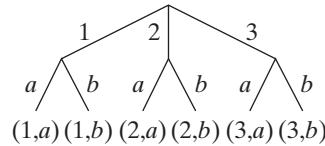
$$Y \times X = \{(a, 1), (b, 1), (a, 2), (b, 2), (a, 3), (b, 3)\}$$

$$X \times X = \{(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3)\}$$

$$Y \times Y = \{(a, a), (a, b), (b, a), (b, b)\}. \quad \blacktriangleleft$$

Example 1.1.26 shows that, in general,  $X \times Y \neq Y \times X$ .

Notice that in Example 1.1.26,  $|X \times Y| = |X| \cdot |Y|$  (both are equal to 6). The reason is that there are 3 ways to choose an element of  $X$  for the first member of the ordered pair, there are 2 ways to choose an element of  $Y$  for the second member of the ordered pair, and  $3 \cdot 2 = 6$  (see Figure 1.1.8). The preceding argument holds for arbitrary finite sets  $X$  and  $Y$ ; it is always true that  $|X \times Y| = |X| \cdot |Y|$ .



**Figure 1.1.8**  $|X \times Y| = |X| \cdot |Y|$ , where  $X = \{1, 2, 3\}$  and  $Y = \{a, b\}$ . There are 3 ways to choose an element of  $X$  for the first member of the ordered pair (shown at the top of the diagram) and, for each of these choices, there are 2 ways to choose an element of  $Y$  for the second member of the ordered pair (shown at the bottom of the diagram). Since there are 3 groups of 2, there are  $3 \cdot 2 = 6$  elements in  $X \times Y$  (labeled at the bottom of the figure).

### Example 1.1.27

A restaurant serves four appetizers,

$$r = \text{ribs}, \quad n = \text{nachos}, \quad s = \text{shrimp}, \quad f = \text{fried cheese},$$

and three entrees,

$$c = \text{chicken}, \quad b = \text{beef}, \quad t = \text{trout}.$$

If we let  $A = \{r, n, s, f\}$  and  $E = \{c, b, t\}$ , the Cartesian product  $A \times E$  lists the 12 possible dinners consisting of one appetizer and one entree. 

Ordered lists need not be restricted to two elements. An ***n*-tuple**, written  $(a_1, a_2, \dots, a_n)$ , takes order into account; that is,

$$(a_1, a_2, \dots, a_n) = (b_1, b_2, \dots, b_n)$$

precisely when

$$a_1 = b_1, a_2 = b_2, \dots, a_n = b_n.$$

The Cartesian product of sets  $X_1, X_2, \dots, X_n$  is defined to be the set of all *n*-tuples  $(x_1, x_2, \dots, x_n)$  where  $x_i \in X_i$  for  $i = 1, \dots, n$ ; it is denoted  $X_1 \times X_2 \times \dots \times X_n$ . 

### Example 1.1.28

If  $X = \{1, 2\}$ ,  $Y = \{a, b\}$ , and  $Z = \{\alpha, \beta\}$ , then

$$\begin{aligned} X \times Y \times Z &= \{(1, a, \alpha), (1, a, \beta), (1, b, \alpha), (1, b, \beta), (2, a, \alpha), (2, a, \beta), \\ &\quad (2, b, \alpha), (2, b, \beta)\}. \end{aligned} \quad \blacktriangleleft$$

Notice that in Example 1.1.28,  $|X \times Y \times Z| = |X| \cdot |Y| \cdot |Z|$ . In general,

$$|X_1 \times X_2 \times \dots \times X_n| = |X_1| \cdot |X_2| \cdots |X_n|.$$

We leave the proof of this last statement as an exercise (see Exercise 27, Section 2.4).

### Example 1.1.29

If  $A$  is a set of appetizers,  $E$  is a set of entrees, and  $D$  is a set of desserts, the Cartesian product  $A \times E \times D$  lists all possible dinners consisting of one appetizer, one entree, and one dessert. 

## 1.1 Problem-Solving Tips

- To verify that two sets  $A$  and  $B$  are equal, written  $A = B$ , show that for every  $x$ , if  $x \in A$ , then  $x \in B$ , and if  $x \in B$ , then  $x \in A$ .
- To verify that two sets  $A$  and  $B$  are *not* equal, written  $A \neq B$ , find at least one element that is in  $A$  but not in  $B$ , or find at least one element that is in  $B$  but not in  $A$ . One or the other conditions suffices; you need not (and may not be able to) show both conditions.
- To verify that  $A$  is a subset of  $B$ , written  $A \subseteq B$ , show that for every  $x$ , if  $x \in A$ , then  $x \in B$ . Notice that if  $A$  is a subset of  $B$ , it is possible that  $A = B$ .
- To verify that  $A$  is *not* a subset of  $B$ , find at least one element that is in  $A$  but not in  $B$ .
- To verify that  $A$  is a proper subset of  $B$ , written  $A \subset B$ , verify that  $A$  is a subset of  $B$  as described previously, and that  $A \neq B$ , that is, that there is at least one element that is in  $B$  but not in  $A$ .
- To visualize relationships among sets, use a Venn diagram. A Venn diagram can suggest whether a statement about sets is true or false.
- A set of elements is determined by its members; order is irrelevant. On the other hand, ordered pairs and  $n$ -tuples take order into account.

## 1.1 Review Exercises

- <sup>†</sup>1. What is a set?
2. What is set notation?
3. Describe the sets  $\mathbf{Z}$ ,  $\mathbf{Q}$ ,  $\mathbf{R}$ ,  $\mathbf{Z}^+$ ,  $\mathbf{Q}^+$ ,  $\mathbf{R}^+$ ,  $\mathbf{Z}^-$ ,  $\mathbf{Q}^-$ ,  $\mathbf{R}^-$ ,  $\mathbf{Z}^{\text{nonneg}}$ ,  $\mathbf{Q}^{\text{nonneg}}$ , and  $\mathbf{R}^{\text{nonneg}}$ , and give two examples of members of each set.
4. If  $X$  is a finite set, what is  $|X|$ ?
5. How do we denote  $x$  is an element of the set  $X$ ?
6. How do we denote  $x$  is not an element of the set  $X$ ?
7. How do we denote the empty set?
8. Define set  $X$  is equal to set  $Y$ . How do we denote  $X$  is equal to  $Y$ ?
9. Explain a method of verifying that sets  $X$  and  $Y$  are equal.
10. Explain a method of verifying that sets  $X$  and  $Y$  are *not* equal.
11. Define  $X$  is a subset of  $Y$ . How do we denote  $X$  is a subset of  $Y$ ?
12. Explain a method of verifying that  $X$  is a subset of  $Y$ .
13. Explain a method of verifying that  $X$  is *not* a subset of  $Y$ .
14. Define  $X$  is a proper subset of  $Y$ . How do we denote  $X$  is a proper subset of  $Y$ ?
15. Explain a method of verifying that  $X$  is a proper subset of  $Y$ .
16. What is the power set of  $X$ ? How is it denoted?
17. Define  $X$  union  $Y$ . How is the union of  $X$  and  $Y$  denoted?
18. If  $S$  is a family of sets, how do we define the union of  $S$ ? How is the union denoted?
19. Define  $X$  intersect  $Y$ . How is the intersection of  $X$  and  $Y$  denoted?
20. If  $S$  is a family of sets, how do we define the intersection of  $S$ ? How is the intersection denoted?
21. Define  $X$  and  $Y$  are disjoint sets.
22. What is a pairwise disjoint family of sets?
23. Define the difference of sets  $X$  and  $Y$ . How is the difference denoted?
24. What is a universal set?
25. What is the complement of the set  $X$ ? How is it denoted?
26. What is a Venn diagram?
27. Draw a Venn diagram of three sets and identify the set represented by each region.

<sup>†</sup>Exercise numbers in color indicate that a hint or solution appears at the back of the book in the section following the References.

## 12 Chapter 1 ◆ Sets and Logic

28. State the associative laws for sets.
29. State the commutative laws for sets.
30. State the distributive laws for sets.
31. State the identity laws for sets.
32. State the complement laws for sets.
33. State the idempotent laws for sets.
34. State the bound laws for sets.
35. State the absorption laws for sets.
36. State the involution law for sets.
37. State the 0/1 laws for sets.
38. State De Morgan's laws for sets.
39. What is a partition of a set  $X$ ?
40. Define the *Cartesian product* of sets  $X$  and  $Y$ . How is this Cartesian product denoted?
41. Define the *Cartesian product* of the sets  $X_1, X_2, \dots, X_n$ . How is this Cartesian product denoted?

### 1.1 Exercises

In Exercises 1–16, let the universe be the set  $U = \{1, 2, 3, \dots, 10\}$ . Let  $A = \{1, 4, 7, 10\}$ ,  $B = \{1, 2, 3, 4, 5\}$ , and  $C = \{2, 4, 6, 8\}$ . List the elements of each set.

1.  $A \cup B$
2.  $B \cap C$
3.  $A - B$
4.  $B - A$
5.  $\bar{A}$
6.  $U - C$
7.  $\bar{U}$
8.  $A \cup \emptyset$
9.  $B \cap \emptyset$
10.  $A \cup U$
11.  $B \cap U$
12.  $A \cap (B \cup C)$
13.  $\bar{B} \cap (C - A)$
14.  $(A \cap B) - C$
15.  $\overline{A \cap B} \cup C$
16.  $(A \cup B) - (C - B)$

In Exercises 17–27, let the universe be the set  $\mathbb{Z}^+$ . Let  $X = \{1, 2, 3, 4, 5\}$  and let  $Y$  be the set of positive, even integers. In set-builder notation,  $Y = \{2n \mid n \in \mathbb{Z}^+\}$ . In Exercises 18–27, give a mathematical notation for the set by listing the elements if the set is finite, by using set-builder notation if the set is infinite, or by using a predefined set such as  $\emptyset$ .

17. Describe  $\bar{Y}$  in words.
18.  $\bar{X}$
19.  $\bar{Y}$
20.  $X \cap Y$
21.  $X \cup Y$
22.  $\bar{X} \cap Y$
23.  $\bar{X} \cup Y$
24.  $X \cap \bar{Y}$
25.  $X \cup \bar{Y}$
26.  $\bar{X} \cap \bar{Y}$
27.  $\bar{X} \cup \bar{Y}$

28. What is the cardinality of  $\emptyset$ ?
29. What is the cardinality of  $\{\emptyset\}$ ?
30. What is the cardinality of  $\{a, b, a, c\}$ ?
31. What is the cardinality of  $\{\{a\}, \{a, b\}, \{a, c\}, a, b\}$ ?

In Exercises 32–35, show, as in Examples 1.1.2 and 1.1.3, that  $A = B$ .

32.  $A = \{3, 2, 1\}$ ,  $B = \{1, 2, 3\}$
33.  $C = \{1, 2, 3\}$ ,  $D = \{2, 3, 4\}$ ,  $A = \{2, 3\}$ ,  $B = C \cap D$

34.  $A = \{1, 2, 3\}$ ,  $B = \{n \mid n \in \mathbb{Z}^+ \text{ and } n^2 < 10\}$
35.  $A = \{x \mid x^2 - 4x + 4 = 1\}$ ,  $B = \{1, 3\}$

In Exercises 36–39, show, as in Example 1.1.4, that  $A \neq B$ .

36.  $A = \{1, 2, 3\}$ ,  $B = \emptyset$
37.  $A = \{1, 2\}$ ,  $B = \{x \mid x^3 - 2x^2 - x + 2 = 0\}$
38.  $A = \{1, 3, 5\}$ ,  $B = \{n \mid n \in \mathbb{Z}^+ \text{ and } n^2 - 1 \leq n\}$
39.  $B = \{1, 2, 3, 4\}$ ,  $C = \{2, 4, 6, 8\}$ ,  $A = B \cap C$

In Exercises 40–43, determine whether each pair of sets is equal.

40.  $\{1, 2, 2, 3\}$ ,  $\{1, 2, 3\}$
41.  $\{1, 1, 3\}$ ,  $\{3, 3, 1\}$
42.  $\{x \mid x^2 + x = 2\}$ ,  $\{1, -1\}$
43.  $\{x \mid x \in \mathbb{R} \text{ and } 0 < x \leq 2\}$ ,  $\{1, 2\}$

In Exercises 44–47, show, as in Examples 1.1.5 and 1.1.6, that  $A \subseteq B$ .

44.  $A = \{1, 2\}$ ,  $B = \{3, 2, 1\}$
45.  $A = \{1, 2\}$ ,  $B = \{x \mid x^3 - 6x^2 + 11x = 6\}$
46.  $A = \{1\} \times \{1, 2\}$ ,  $B = \{1\} \times \{1, 2, 3\}$
47.  $A = \{2n \mid n \in \mathbb{Z}^+\}$ ,  $B = \{n \mid n \in \mathbb{Z}^+\}$

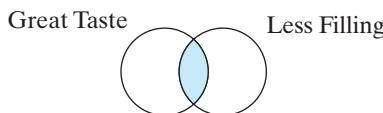
In Exercises 48–51, show, as in Example 1.1.9, that  $A$  is not a subset of  $B$ .

48.  $A = \{1, 2, 3\}$ ,  $B = \{1, 2\}$
49.  $A = \{x \mid x^3 - 2x^2 - x + 2 = 0\}$ ,  $B = \{1, 2\}$
50.  $A = \{1, 2, 3, 4\}$ ,  $C = \{5, 6, 7, 8\}$ ,  $B = \{n \mid n \in A \text{ and } n + m = 8 \text{ for some } m \in C\}$
51.  $A = \{1, 2, 3\}$ ,  $B = \emptyset$

In Exercises 52–59, draw a Venn diagram and shade the given set.

52.  $A \cap \bar{B}$
53.  $\bar{A} - B$
54.  $B \cup (B - A)$
55.  $(A \cup B) - B$
56.  $B \cap \overline{(C \cup A)}$
57.  $(\bar{A} \cup B) \cap (\bar{C} - A)$
58.  $((C \cap A) - (\bar{B} - A)) \cap C$
59.  $(B - \bar{C}) \cup ((B - \bar{A}) \cap (C \cup B))$

60. A television commercial for a popular beverage showed the following Venn diagram



What does the shaded area represent?

*Exercises 61–65 refer to a group of 191 students, of which 10 are taking French, business, and music; 36 are taking French and business; 20 are taking French and music; 18 are taking business and music; 65 are taking French; 76 are taking business; and 63 are taking music.*

61. How many are taking French and music but not business?
62. How many are taking business and neither French nor music?
63. How many are taking French or business (or both)?
64. How many are taking music or French (or both) but not business?
65. How many are taking none of the three subjects?
66. A television poll of 151 persons found that 68 watched “Law and Disorder”; 61 watched “25”; 52 watched “The Tenors”; 16 watched both “Law and Disorder” and “25”; 25 watched both “Law and Disorder” and “The Tenors”; 19 watched both “25” and “The Tenors”; and 26 watched none of these shows. How many persons watched all three shows?
67. In a group of students, each student is taking a mathematics course or a computer science course or both. One-fifth of those taking a mathematics course are also taking a computer science course, and one-eighth of those taking a computer science course are also taking a mathematics course. Are more than one-third of the students taking a mathematics course?

*In Exercises 68–71, let  $X = \{1, 2\}$  and  $Y = \{a, b, c\}$ . List the elements in each set.*

- |                  |                  |
|------------------|------------------|
| 68. $X \times Y$ | 69. $Y \times X$ |
| 70. $X \times X$ | 71. $Y \times Y$ |

*In Exercises 72–75, let  $X = \{1, 2\}$ ,  $Y = \{a\}$ , and  $Z = \{\alpha, \beta\}$ . List the elements of each set.*

- |                           |                                    |
|---------------------------|------------------------------------|
| 72. $X \times Y \times Z$ | 73. $X \times Y \times Y$          |
| 74. $X \times X \times X$ | 75. $Y \times X \times Y \times Z$ |

*In Exercises 76–82, give a geometric description of each set in words. Consider the elements of the sets to be coordinates. For example,  $\mathbf{R} \times \mathbf{Z}$  is the set  $\{(x, n) \mid x \in \mathbf{R} \text{ and } n \in \mathbf{Z}\}$ . Interpreting the ordered pairs  $(x, n)$  as coordinates in the plane, the graph of all*

*such ordered pairs is the set of all parallel horizontal lines spaced one unit apart, one of which passes through  $(0, 0)$ .*

76.  $\mathbf{R} \times \mathbf{R}$
77.  $\mathbf{Z} \times \mathbf{R}$
78.  $\mathbf{R} \times \mathbf{Z}^{\text{nonneg}}$
79.  $\mathbf{Z} \times \mathbf{Z}$
80.  $\mathbf{R} \times \mathbf{R} \times \mathbf{R}$
81.  $\mathbf{R} \times \mathbf{R} \times \mathbf{Z}$
82.  $\mathbf{R} \times \mathbf{Z} \times \mathbf{Z}$

*In Exercises 83–86, list all partitions of the set.*

- |                   |                      |
|-------------------|----------------------|
| 83. $\{1\}$       | 84. $\{1, 2\}$       |
| 85. $\{a, b, c\}$ | 86. $\{a, b, c, d\}$ |

*In Exercises 87–92, answer true or false.*

- |                                             |                                       |
|---------------------------------------------|---------------------------------------|
| 87. $\{x\} \subseteq \{x\}$                 | 88. $\{x\} \in \{x\}$                 |
| 89. $\{x\} \in \{x, \{x\}\}$                | 90. $\{x\} \subseteq \{x, \{x\}\}$    |
| 91. $\{2\} \subseteq \mathcal{P}(\{1, 2\})$ | 92. $\{2\} \in \mathcal{P}(\{1, 2\})$ |

93. List the members of  $\mathcal{P}(\{a, b\})$ . Which are proper subsets of  $\{a, b\}$ ?
94. List the members of  $\mathcal{P}(\{a, b, c, d\})$ . Which are proper subsets of  $\{a, b, c, d\}$ ?
95. If  $X$  has 10 members, how many members does  $\mathcal{P}(X)$  have? How many proper subsets does  $X$  have?
96. If  $X$  has  $n$  members, how many proper subsets does  $X$  have?

*In Exercises 97–100, what relation must hold between sets  $A$  and  $B$  in order for the given condition to be true?*

- |                                       |                                           |
|---------------------------------------|-------------------------------------------|
| 97. $A \cap B = A$                    | 98. $A \cup B = A$                        |
| 99. $\overline{A} \cap U = \emptyset$ | 100. $\overline{A \cap B} = \overline{B}$ |

*The symmetric difference of two sets  $A$  and  $B$  is the set*

$$A \Delta B = (A \cup B) - (A \cap B).$$

101. If  $A = \{1, 2, 3\}$  and  $B = \{2, 3, 4, 5\}$ , find  $A \Delta B$ .
102. Describe the symmetric difference of sets  $A$  and  $B$  in words.
103. Given a universe  $U$ , describe  $A \Delta A$ ,  $A \Delta \overline{A}$ ,  $U \Delta A$ , and  $\emptyset \Delta A$ .
104. Let  $C$  be a circle and let  $\mathcal{D}$  be the set of all diameters of  $C$ . What is  $\cap \mathcal{D}$ ? (Here, by “diameter” we mean a line segment through the center of the circle with its endpoints on the circumference of the circle.)

<sup>†</sup>★105. Let  $P$  denote the set of integers greater than 1. For  $i \geq 2$ , define

$$X_i = \{ik \mid k \in P\}.$$

Describe  $P - \bigcup_{i=2}^{\infty} X_i$ .

<sup>†</sup>A starred exercise indicates a problem of above-average difficulty.

## 1.2 Propositions

---

Which of sentences (a)–(f) are either true or false (but not both)?

- (a) The only positive integers that divide<sup>†</sup> 7 are 1 and 7 itself.
- (b) Alfred Hitchcock won an Academy Award in 1940 for directing *Rebecca*.
- (c) For every positive integer  $n$ , there is a prime number<sup>‡</sup> larger than  $n$ .
- (d) Earth is the only planet in the universe that contains life.
- (e) Buy two tickets to the “Unhinged Universe” rock concert for Friday.
- (f)  $x + 4 = 6$ .

Sentence (a), which is another way to say that 7 is prime, is true.

Sentence (b) is false. Although *Rebecca* won the Academy Award for best picture in 1940, John Ford won the directing award for *The Grapes of Wrath*. It is a surprising fact that Alfred Hitchcock never won an Academy Award for directing.

Sentence (c), which is another way to say that the number of primes is infinite, is true.

Sentence (d) is either true or false (but not both), but no one knows which at this time.

Sentence (e) is neither true nor false [sentence (e) is a command].

The truth of equation (f) depends on the value of the variable  $x$ .

A sentence that is either true or false, but not both, is called a **proposition**. Sentences (a)–(d) are propositions, whereas sentences (e) and (f) are not propositions. A proposition is typically expressed as a declarative sentence (as opposed to a question, command, etc.). Propositions are the basic building blocks of any theory of logic.

We will use variables, such as  $p$ ,  $q$ , and  $r$ , to represent propositions, much as we use letters in algebra to represent numbers. We will also use the notation

$$p: 1 + 1 = 3$$

to define  $p$  to be the proposition  $1 + 1 = 3$ .

In ordinary speech and writing, we combine propositions using connectives such as *and* and *or*. For example, the propositions “It is raining” and “It is cold” can be combined to form the single proposition “It is raining and it is cold.” The formal definitions of *and* and *or* follow.

**Definition 1.2.1** ► Let  $p$  and  $q$  be propositions.

The *conjunction* of  $p$  and  $q$ , denoted  $p \wedge q$ , is the proposition

$$p \text{ and } q.$$

The *disjunction* of  $p$  and  $q$ , denoted  $p \vee q$ , is the proposition

$$p \text{ or } q.$$

### Example 1.2.2

If  $p$ : It is raining, and  $q$ : It is cold, then the conjunction of  $p$  and  $q$  is

$$p \wedge q: \text{It is raining and it is cold.}$$

(1.2.1)

The disjunction of  $p$  and  $q$  is

<sup>†</sup>“Divides” means “divides evenly.” More formally, we say that a nonzero integer  $d$  *divides* an integer  $m$  if there is an integer  $q$  such that  $m = dq$ . We call  $q$  the *quotient*. We will explore the integers in detail in Chapter 5.

<sup>‡</sup>An integer  $n > 1$  is *prime* if the only positive integers that divide  $n$  are 1 and  $n$  itself. For example, 2, 3, and 11 are prime numbers.

$p \vee q$ : It is raining or it is cold. ▶

The truth value of the conjunction  $p \wedge q$  is determined by the truth values of  $p$  and  $q$ , and the definition is based upon the usual interpretation of “and.” Consider the proposition (1.2.1) of Example 1.2.2. If it is raining (i.e.,  $p$  is true) and it is also cold (i.e.,  $q$  is also true), then we would consider the proposition (1.2.1) to be true. However, if it is not raining (i.e.,  $p$  is false) or it is not cold (i.e.,  $q$  is false) or both, then we would consider the proposition (1.2.1) to be false.

The truth values of propositions such as conjunctions and disjunctions can be described by **truth tables**. The truth table of a proposition  $P$  made up of the individual propositions  $p_1, \dots, p_n$  lists all possible combinations of truth values for  $p_1, \dots, p_n$ , T denoting true and F denoting false, and for each such combination lists the truth value of  $P$ . We use a truth table to formally define the truth value of  $p \wedge q$ .

A truth table for a proposition  $P$  made up of  $n$  propositions has  $r = 2^n$  rows. Traditionally, for the first proposition, the first  $r/2$  rows list T and the last  $r/2$  rows list F. The next proposition has  $r/4$  T's alternate with  $r/4$  F's. The next proposition has  $r/8$  T's alternate with  $r/8$  F's, and so on. For example, a proposition  $P$  made up of three propositions  $p_1, p_2$ , and  $p_3$  has  $8 = 2^3$  rows. Proposition  $p_1$  will list 4 =  $8/2$  T's followed by 4 F's. Proposition  $p_2$  will list 2 =  $8/4$  T's followed by 2 F's, followed by 2 T's, followed by 2 F's. Proposition  $p_3$  will have one T, followed by one F, followed by one T, and so on. The truth table without the truth values of  $P$  would be

| $p_1$ | $p_2$ | $p_3$ | $P$ |
|-------|-------|-------|-----|
| T     | T     | T     |     |
| T     | T     | F     |     |
| T     | F     | T     |     |
| T     | F     | F     |     |
| F     | T     | T     |     |
| F     | T     | F     |     |
| F     | F     | T     |     |
| F     | F     | F     |     |

Here is where the  
truth values of  $P$  go.

Notice that all possible combinations of truth values for  $p_1, p_2, p_3$  are listed.

**Definition 1.2.3** ► The truth value of the proposition  $p \wedge q$  is defined by the truth table

| $p$ | $q$ | $p \wedge q$ |
|-----|-----|--------------|
| T   | T   | T            |
| T   | F   | F            |
| F   | T   | F            |
| F   | F   | F            |

Definition 1.2.3 states that the conjunction  $p \wedge q$  is true provided that  $p$  and  $q$  are both true;  $p \wedge q$  is false otherwise. ▶

#### Example 1.2.4

If  $p$ : A decade is 10 years, and  $q$ : A millennium is 100 years, then  $p$  is true,  $q$  is false (a millennium is 1000 years), and the conjunction,

$p \wedge q$ : A decade is 10 years and a millennium is 100 years, ▶

is false.

**Example 1.2.5**

Most programming languages define “and” exactly as in Definition 1.2.3. For example, in the Java programming language, (logical) “and” is denoted `&&`, and the expression

$$x < 10 \text{ } \&\& \text{ } y > 4$$

is true precisely when the value of the variable  $x$  is less than 10 (i.e.,  $x < 10$  is true) and the value of the variable  $y$  is greater than 4 (i.e.,  $y > 4$  is also true). 

The truth value of the disjunction  $p \vee q$  is also determined by the truth values of  $p$  and  $q$ , and the definition is based upon the “inclusive” interpretation of “or.” Consider the proposition,

$$p \vee q: \text{It is raining or it is cold,} \quad (1.2.2)$$

of Example 1.2.2. If it is raining (i.e.,  $p$  is true) or it is cold (i.e.,  $q$  is also true) *or both*, then we would consider the proposition (1.2.2) to be true (i.e.,  $p \vee q$  is true). If it is not raining (i.e.,  $p$  is false) and it is not cold (i.e.,  $q$  is also false), then we would consider the proposition (1.2.2) to be false (i.e.,  $p \vee q$  is false). The **inclusive-or** of propositions  $p$  and  $q$  is true if  $p$  or  $q$ , *or both*, is true, and false otherwise. There is also an **exclusive-or** (see Exercise 67) that defines  $p$  *exor*  $q$  to be true if  $p$  or  $q$ , *but not both*, is true, and false otherwise.

**Definition 1.2.6** ► The truth value of the proposition  $p \vee q$ , called the *inclusive-or* of  $p$  and  $q$ , is defined by the truth table

| $p$ | $q$ | $p \vee q$ |
|-----|-----|------------|
| T   | T   | T          |
| T   | F   | T          |
| F   | T   | T          |
| F   | F   | F          |

**Example 1.2.7**

If  $p$ : A millennium is 100 years, and  $q$ : A millennium is 1000 years, then  $p$  is false,  $q$  is true, and the disjunction,

$$p \vee q: \text{A millennium is 100 years or a millennium is 1000 years,}$$

is true. 

**Example 1.2.8**

Most programming languages define (inclusive) “or” exactly as in Definition 1.2.6. For example, in the Java programming language, (logical) “or” is denoted `||`, and the expression

$$x < 10 \text{ } || \text{ } y > 4$$

is true precisely when the value of the variable  $x$  is less than 10 (i.e.,  $x < 10$  is true) or the value of the variable  $y$  is greater than 4 (i.e.,  $y > 4$  is true) or both. 

In ordinary language, propositions being combined (e.g.,  $p$  and  $q$  combined to give the proposition  $p \vee q$ ) are normally related; but in logic, these propositions are not required to refer to the same subject matter. For example, in logic, we permit propositions such as

$$3 < 5 \text{ or Paris is the capital of England.}$$

Logic is concerned with the form of propositions and the relation of propositions to each other and not with the subject matter itself. (The given proposition is true because  $3 < 5$  is true.)

The final operator on a proposition  $p$  that we discuss in this section is the **negation** of  $p$ .

**Definition 1.2.9** ► The *negation* of  $p$ , denoted  $\neg p$ , is the proposition

not  $p$ .

The truth value of the proposition  $\neg p$  is defined by the truth table

| $p$ | $\neg p$ |
|-----|----------|
| T   | F        |
| F   | T        |

In English, we sometimes write  $\neg p$  as “It is not the case that  $p$ .” For example, if

$p$ : Paris is the capital of England,

the negation of  $p$  could be written

$\neg p$ : It is not the case that Paris is the capital of England,

or more simply as

$\neg p$ : Paris is not the capital of England.

### Example 1.2.10

If

$p$ :  $\pi$  was calculated to 1,000,000 decimal digits in 1954,

the negation of  $p$  is the proposition

$\neg p$ :  $\pi$  was not calculated to 1,000,000 decimal digits in 1954.

It was not until 1973 that 1,000,000 decimal digits of  $\pi$  were computed; so,  $p$  is false. (Since then over 12 trillion decimal digits of  $\pi$  have been computed.) Since  $p$  is false,  $\neg p$  is true.

### Example 1.2.11

Most programming languages define “not” exactly as in Definition 1.2.9. For example, in the Java programming language, “not” is denoted `!`, and the expression

`!(x < 10)`

is true precisely when the value of the variable  $x$  is not less than 10 (i.e.,  $x$  is greater than or equal to 10).

In expressions involving some or all of the operators  $\neg$ ,  $\wedge$ , and  $\vee$ , in the absence of parentheses, we first evaluate  $\neg$ , then  $\wedge$ , and then  $\vee$ . We call such a convention **operator precedence**. In algebra, operator precedence tells us to evaluate  $\cdot$  and  $/$  before  $+$  and  $-$ .

### Example 1.2.12

Given that proposition  $p$  is false, proposition  $q$  is true, and proposition  $r$  is false, determine whether the proposition  $\neg p \vee q \wedge r$  is true or false.

**SOLUTION** We first evaluate  $\neg p$ , which is true. We next evaluate  $q \wedge r$ , which is false. Finally, we evaluate  $\neg p \vee q \wedge r$ , which is true. 

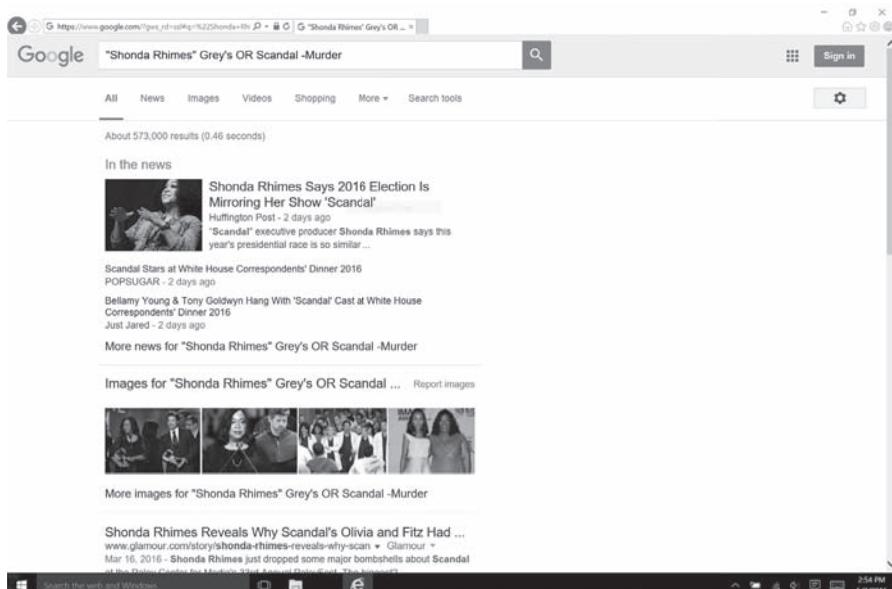
### Example 1.2.13

**Searching the Web** A variety of Web search engines are available (e.g., Google, Yahoo!, Baidu) that allow the user to enter keywords that the search engine then tries to match with Web pages. For example, entering *mathematics* produces a (huge!) list of pages that contain the word “mathematics.” Some search engines allow the user to use *and*, *or*, and *not* operators to combine keywords (see Figure 1.2.1), thus allowing more complex searches. In the Google search engine, *and* is the default operator so that, for example, entering *discrete mathematics* produces a list of pages containing both of the words “discrete” and “mathematics.” The *or* operator is OR, and the *not* operator is the minus sign  $-$ . Furthermore, enclosing a phrase, typically with embedded spaces, in double quotation marks causes the phrase to be treated as a single word. For example, to search for pages containing the keywords

“Shonda Rhimes” and (*Grey’s* or *Scandal*) and (not *Murder*),

the user could enter

“Shonda Rhimes” Grey’s OR Scandal -Murder

Google and the Google logo are registered trademarks of Google Inc., used with permission.

**Figure 1.2.1** The Google search engine, which allows the user to use *and* (space), *or* (OR), and *not*  $(-)$  operators to combine keywords. As shown, Google found about 573,000 Web pages containing “Shonda Rhimes” and (*Grey’s* or *Scandal*) and (not *Murder*). 

## 1.2 Problem-Solving Tips

Although there may be a shorter way to determine the truth values of a proposition  $P$  formed by combining propositions  $p_1, \dots, p_n$  using operators such as  $\neg$  and  $\vee$ , a truth table will always supply all possible truth values of  $P$  for various truth values of the constituent propositions  $p_1, \dots, p_n$ .

## 1.2 Review Exercises

1. What is a proposition?
2. What is a truth table?
3. What is the conjunction of  $p$  and  $q$ ? How is it denoted?
4. Give the truth table for the conjunction of  $p$  and  $q$ .
5. What is the disjunction of  $p$  and  $q$ ? How is it denoted?
6. Give the truth table for the disjunction of  $p$  and  $q$ .
7. What is the negation of  $p$ ? How is it denoted?
8. Give the truth table for the negation of  $p$ .

## 1.2 Exercises

Determine whether each sentence in Exercises 1–12 is a proposition. If the sentence is a proposition, write its negation. (You are not being asked for the truth values of the sentences that are propositions.)

1.  $2 + 5 = 19$ .
2.  $6 + 9 = 15$ .
3.  $x + 9 = 15$ .
4.  $\pi = 3.14$
5. Waiter, will you serve the nuts—I mean, would you serve the guests the nuts?
6. For some positive integer  $n$ ,  $19340 = n \cdot 17$ .
7. Audrey Meadows was the original “Alice” in “The Honeymooners.”
8. Peel me a grape.
9. The line “Play it again, Sam” occurs in the movie *Casablanca*.
10. Every even integer greater than 4 is the sum of two primes.
11. The difference of two primes.
- ★12. This statement is false.

Exercises 13–16 refer to a coin that is flipped 10 times. Write the negation of the proposition.

13. Ten heads were obtained.
14. Some heads were obtained.
15. Some heads and some tails were obtained.
16. At least one head was obtained.

Given that proposition  $p$  is false, proposition  $q$  is true, and proposition  $r$  is false, determine whether each proposition in Exercises 17–22 is true or false.

17.  $p \vee q$
18.  $\neg p \vee \neg q$
19.  $\neg p \vee q$
20.  $\neg p \vee \neg(q \wedge r)$
21.  $\neg(p \vee q) \wedge (\neg p \vee r)$
22.  $(p \vee \neg r) \wedge \neg((q \vee r) \vee \neg(r \vee p))$

Write the truth table of each proposition in Exercises 23–30.

23.  $p \wedge \neg q$
24.  $(\neg p \vee \neg q) \vee p$
25.  $(p \vee q) \wedge \neg p$
26.  $(p \wedge q) \wedge \neg p$
27.  $(p \wedge q) \vee (\neg p \vee q)$
28.  $\neg(p \wedge q) \vee (r \wedge \neg p)$
29.  $(p \vee q) \wedge (\neg p \vee q) \wedge (p \vee \neg q) \wedge (\neg p \vee \neg q)$
30.  $\neg(p \wedge q) \vee (\neg q \vee r)$

In Exercises 31–33, represent the given proposition symbolically by letting

$$p: 5 < 9, \quad q: 9 < 7, \quad r: 5 < 7.$$

Determine whether each proposition is true or false.

31.  $5 < 9$  and  $9 < 7$ .
32. It is not the case that  $(5 < 9$  and  $9 < 7)$ .
33.  $5 < 9$  or it is not the case that  $(9 < 7$  and  $5 < 7)$ .

In Exercises 34–39, formulate the symbolic expression in words using

$$\begin{aligned} p &: \text{Lee takes computer science.} \\ q &: \text{Lee takes mathematics.} \end{aligned}$$

34.  $\neg p$
35.  $p \wedge q$
36.  $p \vee q$
37.  $p \vee \neg q$
38.  $p \wedge \neg q$
39.  $\neg p \wedge \neg q$

In Exercises 40–44, formulate the symbolic expression in words using

$$\begin{aligned} p &: \text{You play football.} \\ q &: \text{You miss the midterm exam.} \\ r &: \text{You pass the course.} \end{aligned}$$

40.  $p \wedge q$
41.  $\neg q \wedge r$
42.  $p \vee q \vee r$
43.  $\neg(p \vee q) \vee r$
44.  $(p \wedge q) \vee (\neg q \wedge r)$

In Exercises 45–49, formulate the symbolic expression in words using

$$\begin{aligned} p &: \text{Today is Monday.} \\ q &: \text{It is raining.} \\ r &: \text{It is hot.} \end{aligned}$$

45.  $p \vee q$
46.  $\neg p \wedge (q \vee r)$
47.  $\neg(p \vee q) \wedge r$
48.  $(p \wedge q) \wedge \neg(r \vee p)$
49.  $(p \wedge (q \vee r)) \wedge (r \vee (q \vee p))$

In Exercises 50–55, represent the proposition symbolically by letting

$$\begin{aligned} p &: \text{There is a hurricane.} \\ q &: \text{It is raining.} \end{aligned}$$

50. There is no hurricane.

51. There is a hurricane and it is raining.  
 52. There is a hurricane, but it is not raining.  
 53. There is no hurricane and it is not raining.  
 54. Either there is a hurricane or it is raining (or both).  
 55. Either there is a hurricane or it is raining, but there is no hurricane.

*In Exercises 56–60, represent the proposition symbolically by letting*

$$\begin{aligned} p &: \text{You run 10 laps daily.} \\ q &: \text{You are healthy.} \\ r &: \text{You take multi-vitamins.} \end{aligned}$$

56. You run 10 laps daily, but you are not healthy.  
 57. You run 10 laps daily, you take multi-vitamins, and you are healthy.  
 58. You run 10 laps daily or you take multi-vitamins, and you are healthy.  
 59. You do not run 10 laps daily, you do not take multi-vitamins, and you are not healthy.  
 60. Either you are healthy or you do not run 10 laps daily, and you do not take multi-vitamins.

*In Exercises 61–66, represent the proposition symbolically by letting*

$$\begin{aligned} p &: \text{You heard the "Flying Pigs" rock concert.} \\ q &: \text{You heard the "Y2K" rock concert.} \\ r &: \text{You have sore eardrums.} \end{aligned}$$

61. You heard the “Flying Pigs” rock concert, and you have sore eardrums.  
 62. You heard the “Flying Pigs” rock concert, but you do not have sore eardrums.  
 63. You heard the “Flying Pigs” rock concert, you heard the “Y2K” rock concert, and you have sore eardrums.

64. You heard either the “Flying Pigs” rock concert or the “Y2K” rock concert, but you do not have sore eardrums.  
 65. You did not hear the “Flying Pigs” rock concert and you did not hear the “Y2K” rock concert, but you have sore eardrums.  
 66. It is not the case that: You heard the “Flying Pigs” rock concert or you heard the “Y2K” rock concert or you do not have sore eardrums.  
 67. Give the truth table for the exclusive-or of  $p$  and  $q$  in which  $p$  exor  $q$  is true if either  $p$  or  $q$ , but not both, is true.

*In Exercises 68–74, state the meaning of each sentence if “or” is interpreted as the inclusive-or; then, state the meaning of each sentence if “or” is interpreted as the exclusive-or (see Exercise 67). In each case, which meaning do you think is intended?*

68. To enter Utopia, you must show a driver’s license or a passport.  
 69. To enter Utopia, you must possess a driver’s license or a passport.  
 70. The prerequisite to data structures is a course in Java or C++.  
 71. The car comes with a cupholder that heats or cools your drink.  
 72. We offer \$1000 cash or 0 percent interest for two years.  
 73. Do you want fries or a salad with your burger?  
 74. The meeting will be canceled if fewer than 10 persons sign up or at least 3 inches of snow falls.  
 75. At one time, the following ordinance was in effect in Naperville, Illinois: “It shall be unlawful for any person to keep more than three [3] dogs and three [3] cats upon his property within the city.” Was Charles Marko, who owned five dogs and no cats, in violation of the ordinance? Explain.  
 76. Write a command to search the Web for national parks in North or South Dakota.  
 77. Write a command to search the Web for information on lung disease other than cancer.  
 78. Write a command to search the Web for minor league baseball teams in Illinois that are not in the Midwest League.

## 1.3 Conditional Propositions and Logical Equivalence

The dean has announced that

If the Mathematics Department gets an additional \$60,000,  
 then it will hire one new faculty member. (1.3.1)

Statement (1.3.1) states that on the condition that the Mathematics Department gets an additional \$60,000, then the Mathematics Department will hire one new faculty member. A proposition such as (1.3.1) is called a **conditional proposition**.

**Definition 1.3.1** ► If  $p$  and  $q$  are propositions, the proposition

$$\text{if } p \text{ then } q \quad \text{(1.3.2)}$$

is called a *conditional proposition* and is denoted

$$p \rightarrow q.$$

The proposition  $p$  is called the *hypothesis* (or *antecedent*), and the proposition  $q$  is called the *conclusion* (or *consequent*). 

### Example 1.3.2

If we define

- $p$ : The Mathematics Department gets an additional \$60,000,
- $q$ : The Mathematics Department will hire one new faculty member,

then proposition (1.3.1) assumes the form (1.3.2). The hypothesis is the statement “The Mathematics Department gets an additional \$60,000,” and the conclusion is the statement “The Mathematics Department will hire one new faculty member.” 

What is the truth value of the dean’s statement (1.3.1)? First, suppose that the Mathematics Department gets an additional \$60,000. If the Mathematics Department does hire an additional faculty member, surely the dean’s statement is true. (Using the notation of Example 1.3.2, if  $p$  and  $q$  are both true, then  $p \rightarrow q$  is true.) On the other hand, if the Mathematics Department gets an additional \$60,000 and does *not* hire an additional faculty member, the dean is wrong—statement (1.3.1) is false. (If  $p$  is true and  $q$  is false, then  $p \rightarrow q$  is false.) Now, suppose that the Mathematics Department does *not* get an additional \$60,000. In this case, the Mathematics Department might or might not hire an additional faculty member. (Perhaps a member of the department retires and someone is hired to replace the retiree. On the other hand, the department might not hire anyone.) Surely we would not consider the dean’s statement to be false. Thus, if the Mathematics Department does *not* get an additional \$60,000, the dean’s statement must be true, regardless of whether the department hires an additional faculty member or not. (If  $p$  is false, then  $p \rightarrow q$  is true whether  $q$  is true or false.) This discussion motivates the following definition.

**Definition 1.3.3** ► The truth value of the conditional proposition  $p \rightarrow q$  is defined by the following truth table:

| $p$ | $q$ | $p \rightarrow q$ |
|-----|-----|-------------------|
| T   | T   | T                 |
| T   | F   | F                 |
| F   | T   | T                 |
| F   | F   | T                 |



For those who need additional evidence that we should define  $p \rightarrow q$  to be true when  $p$  is false, we offer further justification. Most people would agree that the proposition,

$$\text{For all real numbers } x, \text{ if } x > 0, \text{ then } x^2 > 0, \quad (1.3.3)$$

is true. (In Section 1.5, we will discuss such “for all” statements formally and in detail.) In the following discussion, we let  $P(x)$  denote  $x > 0$  and  $Q(x)$  denote  $x^2 > 0$ . That proposition (1.3.3) is true means that no matter which real number we replace  $x$  with, the proposition

$$\text{if } P(x) \text{ then } Q(x) \quad (1.3.4)$$

that results is true. For example, if  $x = 3$ , then  $P(3)$  and  $Q(3)$  are both true ( $3 > 0$  and  $3^2 > 0$  are both true), and, by Definition 1.3.3, (1.3.4) is true. Now let us consider the situation when  $P(x)$  is false. If  $x = -2$ , then  $P(-2)$  is false ( $-2 > 0$  is false) and  $Q(-2)$  is true [ $(-2)^2 > 0$  is true]. In order for proposition (1.3.4) to be true in this case, we must define  $p \rightarrow q$  to be true when  $p$  is false and  $q$  is true. This is exactly what occurs in the third line of the truth table of Definition 1.3.3. If  $x = 0$ , then  $P(0)$  and  $Q(0)$  are both false ( $0 > 0$  and  $0^2 > 0$  are both false). In order for proposition (1.3.4) to be true in this case, we must define  $p \rightarrow q$  to be true when both  $p$  and  $q$  are false. This is exactly what occurs in the fourth line of the truth table of Definition 1.3.3. Even more motivation for defining  $p \rightarrow q$  to be true when  $p$  is false is given in Exercises 77 and 78.

**Example 1.3.4**

Let  $p : 1 > 2$  and  $q : 4 < 8$ . Then  $p$  is false and  $q$  is true. Therefore,  $p \rightarrow q$  is true and  $q \rightarrow p$  is false. 

In expressions that involve the logical operators  $\wedge$ ,  $\vee$ ,  $\neg$ , and  $\rightarrow$ , the conditional operator  $\rightarrow$  is evaluated last. For example,

$$p \vee q \rightarrow \neg r$$

is interpreted as

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

**Example 1.3.5**

Assuming that  $p$  is true,  $q$  is false, and  $r$  is true, find the truth value of each proposition.

- (a)  $p \wedge q \rightarrow r$
- (b)  $p \vee q \rightarrow \neg r$
- (c)  $p \wedge (q \rightarrow r)$
- (d)  $p \rightarrow (q \rightarrow r)$

**SOLUTION**

- (a) We first evaluate  $p \wedge q$  because  $\rightarrow$  is evaluated last. Since  $p$  is true and  $q$  is false,  $p \wedge q$  is false. Therefore,  $p \wedge q \rightarrow r$  is true (regardless of whether  $r$  is true or false).
- (b) We first evaluate  $\neg r$ . Since  $r$  is true,  $\neg r$  is false. We next evaluate  $p \vee q$ . Since  $p$  is true and  $q$  is false,  $p \vee q$  is true. Therefore,  $p \vee q \rightarrow \neg r$  is false.
- (c) Since  $q$  is false,  $q \rightarrow r$  is true (regardless of whether  $r$  is true or false). Since  $p$  is true,  $p \wedge (q \rightarrow r)$  is true.
- (d) Since  $q$  is false,  $q \rightarrow r$  is true (regardless of whether  $r$  is true or false). Thus,  $p \rightarrow (q \rightarrow r)$  is true (regardless of whether  $p$  is true or false). 

A conditional proposition that is true because the hypothesis is false is said to be **true by default** or **vacuously true**. For example, if the proposition,

If the Mathematics Department gets an additional \$60,000, then it will hire one new faculty member,

is true because the Mathematics Department did not get an additional \$60,000, we would say that the proposition is true by default or that it is vacuously true.

Some statements not of the form (1.3.2) may be rephrased as conditional propositions, as the next example illustrates.

**Example 1.3.6**

Restate each proposition in the form (1.3.2) of a conditional proposition.

- (a) Mary will be a good student if she studies hard.
- (b) John takes calculus only if he has sophomore, junior, or senior standing.
- (c) When you sing, my ears hurt.
- (d) A necessary condition for the Cubs to win the World Series is that they sign a right-handed relief pitcher.
- (e) A sufficient condition for Maria to visit France is that she goes to the Eiffel Tower.

**SOLUTION**

- (a) The hypothesis is the clause following *if*; thus an equivalent formulation is

If Mary studies hard, then she will be a good student.

- (b) The statement means that in order for John to take calculus, he must have sophomore, junior, or senior standing. In particular, if he is a freshman, he may *not* take calculus. Thus, we can conclude that if he takes calculus, then he has sophomore, junior, or senior standing. Therefore an equivalent formulation is

If John takes calculus, then he has sophomore, junior, or senior standing.

Notice that

If John has sophomore, junior, or senior standing, then he takes calculus,

is *not* an equivalent formulation. If John has sophomore, junior, or senior standing, he may or may *not* take calculus. (Although eligible to take calculus, he may have decided not to.)

The “*if p then q*” formulation emphasizes the hypothesis, whereas the “*p only if q*” formulation emphasizes the conclusion; the difference is only stylistic.

- (c) *When* means the same as *if*; thus an equivalent formulation is

If you sing, then my ears hurt.

- (d) A **necessary condition** is just that: a condition that is *necessary* for a particular outcome to be achieved. The condition does *not* guarantee the outcome; but, if the condition does not hold, the outcome will not be achieved. Here, the given statement means that if the Cubs win the World Series, we can be sure that they signed a right-handed relief pitcher since, without such a signing, they would not have won the World Series. Thus, an equivalent formulation of the given statement is

If the Cubs win the World Series, then they signed a right-handed relief pitcher.

The conclusion expresses a necessary condition.

Notice that

If the Cubs sign a right-handed relief pitcher, then they win the World Series,

is *not* an equivalent formulation. Signing a right-handed relief pitcher does not guarantee a World Series win. However, *not* signing a right-handed relief pitcher guarantees that they will not win the World Series.

- (e) Similarly, a **sufficient condition** is a condition that *suffices* to guarantee a particular outcome. If the condition does not hold, the outcome might be achieved in other ways or it might not be achieved at all; but if the condition does hold, the outcome is guaranteed. Here, to be sure that Maria visits France, it suffices for her to go to the Eiffel Tower. (There are surely other ways to ensure that Maria visits France; for example, she could go to Lyon.) Thus, an equivalent formulation of the given statement is

If Maria goes to the Eiffel Tower, then she visits France.

The hypothesis expresses a sufficient condition.

Notice that

If Maria visits France, then she goes to the Eiffel Tower,

is *not* an equivalent formulation. As we have already noted, there are ways other than going to the Eiffel Tower to ensure that Maria visits France. 

Example 1.3.4 shows that the proposition  $p \rightarrow q$  can be true while the proposition  $q \rightarrow p$  is false. We call the proposition  $q \rightarrow p$  the **converse** of the proposition  $p \rightarrow q$ . Thus a conditional proposition can be true while its converse is false.

### Example 1.3.7

Write the conditional proposition,

If Jerry receives a scholarship, then he will go to college,

and its converse symbolically and in words. Also, assuming that Jerry does not receive a scholarship, but wins the lottery and goes to college anyway, find the truth value of the original proposition and its converse.

**SOLUTION** Let  $p$ : Jerry receives a scholarship, and  $q$ : Jerry goes to college. The given proposition can be written symbolically as  $p \rightarrow q$ . Since the hypothesis  $p$  is false, the conditional proposition is true.

The converse of the proposition is

If Jerry goes to college, then he receives a scholarship.

The converse can be written symbolically as  $q \rightarrow p$ . Since the hypothesis  $q$  is true and the conclusion  $p$  is false, the converse is false. 

Another useful proposition is

$p$  if and only if  $q$ ,

which is considered to be true precisely when  $p$  and  $q$  have the same truth values (i.e.,  $p$  and  $q$  are both true or  $p$  and  $q$  are both false).

**Definition 1.3.8** ► If  $p$  and  $q$  are propositions, the proposition

$p$  if and only if  $q$

is called a *biconditional proposition* and is denoted

$$p \leftrightarrow q.$$

The truth value of the proposition  $p \leftrightarrow q$  is defined by the following truth table:

| $p$ | $q$ | $p \leftrightarrow q$ |
|-----|-----|-----------------------|
| T   | T   | T                     |
| T   | F   | F                     |
| F   | T   | F                     |
| F   | F   | T                     |

It is traditional in mathematical definitions to use “if” to mean “if and only if.” Consider, for example, the definition of set equality: If sets  $X$  and  $Y$  have the same elements, then  $X$  and  $Y$  are equal. The meaning of this definition is that sets  $X$  and  $Y$  have the same elements *if and only if*  $X$  and  $Y$  are equal.

An alternative way to state “ $p$  if and only if  $q$ ” is “ $p$  is a necessary and sufficient condition for  $q$ .” The proposition “ $p$  if and only if  $q$ ” is sometimes written “ $p$  iff  $q$ .”

### Example 1.3.9

The proposition

$$1 < 5 \text{ if and only if } 2 < 8 \quad (1.3.5)$$

can be written symbolically as  $p \leftrightarrow q$  if we define  $p : 1 < 5$  and  $q : 2 < 8$ . Since both  $p$  and  $q$  are true, the proposition  $p \leftrightarrow q$  is true.

An alternative way to state (1.3.5) is: A necessary and sufficient condition for  $1 < 5$  is that  $2 < 8$ .

In some cases, two different propositions have the same truth values no matter what truth values their constituent propositions have. Such propositions are said to be **logically equivalent**.

**Definition 1.3.10** ► Suppose that the propositions  $P$  and  $Q$  are made up of the propositions  $p_1, \dots, p_n$ . We say that  $P$  and  $Q$  are *logically equivalent* and write

$$P \equiv Q,$$

provided that, given any truth values of  $p_1, \dots, p_n$ , either  $P$  and  $Q$  are both true, or  $P$  and  $Q$  are both false.

### Example 1.3.11

**De Morgan's Laws for Logic** We will verify the first of **De Morgan's laws**

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

and leave the second as an exercise (see Exercise 79).

By writing the truth tables for  $P = \neg(p \vee q)$  and  $Q = \neg p \wedge \neg q$ , we can verify that, given any truth values of  $p$  and  $q$ , either  $P$  and  $Q$  are both true or  $P$  and  $Q$  are both false:

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

Thus  $P$  and  $Q$  are logically equivalent.

**Example 1.3.12** Show that, in Java, the expressions
$$x < 10 \quad \text{and} \quad x > 20$$

and

$$\neg(x \geq 10 \quad \&\quad x \leq 20)$$
are equivalent. (In Java,  $\geq$  means  $\geq$ , and  $\leq$  means  $\leq$ .)

**SOLUTION** If we let  $p$  denote the expression  $x \geq 10$  and  $q$  denote the expression  $x \leq 20$ , the expression  $\neg(x \geq 10 \quad \&\quad x \leq 20)$  becomes  $\neg(p \wedge q)$ . By De Morgan's second law,  $\neg(p \wedge q)$  is equivalent to  $\neg p \vee \neg q$ . Since  $\neg p$  translates as  $x < 10$  and  $\neg q$  translates as  $x > 20$ ,  $\neg p \vee \neg q$  translates as  $x < 10 \quad \text{or} \quad x > 20$ . Therefore, the expressions  $x < 10 \quad \text{or} \quad x > 20$  and  $\neg(x \geq 10 \quad \&\quad x \leq 20)$  are equivalent. ▲

Our next example gives a logically equivalent form of the negation of  $p \rightarrow q$ .**Example 1.3.13** Show that the negation of  $p \rightarrow q$  is logically equivalent to  $p \wedge \neg q$ .

**SOLUTION** By writing the truth tables for  $P = \neg(p \rightarrow q)$  and  $Q = p \wedge \neg q$ , we can verify that, given any truth values of  $p$  and  $q$ , either  $P$  and  $Q$  are both true or  $P$  and  $Q$  are both false:

| $p$ | $q$ | $\neg(p \rightarrow q)$ | $p \wedge \neg q$ |
|-----|-----|-------------------------|-------------------|
| T   | T   | F                       | F                 |
| T   | F   | T                       | T                 |
| F   | T   | F                       | F                 |
| F   | F   | F                       | F                 |

Thus  $P$  and  $Q$  are logically equivalent. ▲**Example 1.3.14** Use the logical equivalence of  $\neg(p \rightarrow q)$  and  $p \wedge \neg q$  (see Example 1.3.13) to write the negation of

If Jerry receives a scholarship, then he goes to college,  
symbolically and in words.

**SOLUTION** We let  $p$ : Jerry receives a scholarship, and  $q$ : Jerry goes to college. The given proposition can be written symbolically as  $p \rightarrow q$ . Its negation is logically equivalent to  $p \wedge \neg q$ . In words, this last expression is

Jerry receives a scholarship and he does not go to college. ▲

We now show that, according to our definitions,  $p \leftrightarrow q$  is logically equivalent to  $p \rightarrow q$  and  $q \rightarrow p$ . In words,

 $p$  if and only if  $q$ 

is logically equivalent to

if  $p$  then  $q$  and if  $q$  then  $p$ .

**Example 1.3.15** The truth table shows that

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

| $p$ | $q$ | $p \leftrightarrow q$ | $p \rightarrow q$ | $q \rightarrow p$ | $(p \rightarrow q) \wedge (q \rightarrow p)$ |
|-----|-----|-----------------------|-------------------|-------------------|----------------------------------------------|
| T   | T   | T                     | T                 | T                 | T                                            |
| T   | F   | F                     | F                 | T                 | F                                            |
| F   | T   | F                     | T                 | F                 | F                                            |
| F   | F   | T                     | T                 | T                 | T                                            |

Consider again the definition of set equality: If sets  $X$  and  $Y$  have the same elements, then  $X$  and  $Y$  are equal. We noted that the meaning of this definition is that sets  $X$  and  $Y$  have the same elements if and only if  $X$  and  $Y$  are equal. Example 1.3.15 shows that an equivalent formulation is: If sets  $X$  and  $Y$  have the same elements, then  $X$  and  $Y$  are equal, and if  $X$  and  $Y$  are equal, then  $X$  and  $Y$  have the same elements.

We conclude this section by defining the **contrapositive** of a conditional proposition. We will see (in Theorem 1.3.18) that the contrapositive is an alternative, logically equivalent form of the conditional proposition. Exercise 80 gives another logically equivalent form of the conditional proposition.

**Definition 1.3.16** ► The *contrapositive* (or *transposition*) of the conditional proposition  $p \rightarrow q$  is the proposition  $\neg q \rightarrow \neg p$ .

Notice the difference between the contrapositive and the converse. The converse of a conditional proposition merely reverses the roles of  $p$  and  $q$ , whereas the contrapositive reverses the roles of  $p$  and  $q$  and negates each of them.

**Example 1.3.17** Write the conditional proposition,

If the network is down, then Dale cannot access the internet,

symbolically. Write the contrapositive and the converse symbolically and in words. Also, assuming that the network is not down and Dale can access the internet, find the truth value of the original proposition, its contrapositive, and its converse.

**SOLUTION** Let  $p$ : The network is down, and  $q$ : Dale cannot access the internet. The given proposition can be written symbolically as  $p \rightarrow q$ . Since the hypothesis  $p$  is false, the conditional proposition is true.

The contrapositive can be written symbolically as  $\neg q \rightarrow \neg p$  and, in words,

If Dale can access the internet, then the network is not down.

Since the hypothesis  $\neg q$  and conclusion  $\neg p$  are both true, the contrapositive is true. (Theorem 1.3.18 will show that the conditional proposition and its contrapositive are logically equivalent, that is, that they always have the same truth value.)

The converse of the given proposition can be written symbolically as  $q \rightarrow p$  and, in words,

If Dale cannot access the internet, then the network is down.

Since the hypothesis  $q$  is false, the converse is true.

An important fact is that a conditional proposition and its contrapositive are logically equivalent.

**Theorem 1.3.18**

The conditional proposition  $p \rightarrow q$  and its contrapositive  $\neg q \rightarrow \neg p$  are logically equivalent.

**Proof** The truth table

| $p$ | $q$ | $p \rightarrow q$ | $\neg q \rightarrow \neg p$ |
|-----|-----|-------------------|-----------------------------|
| T   | T   | T                 | T                           |
| T   | F   | F                 | F                           |
| F   | T   | T                 | T                           |
| F   | F   | T                 | T                           |

shows that  $p \rightarrow q$  and  $\neg q \rightarrow \neg p$  are logically equivalent. ◀

In ordinary language, “if” is often used to mean “if and only if.” Consider the statement

If you fix my computer, then I’ll pay you \$50.

The intended meaning is

If you fix my computer, then I’ll pay you \$50, and  
if you do not fix my computer, then I will not pay you \$50,

which is logically equivalent to (see Theorem 1.3.18)

If you fix my computer, then I’ll pay you \$50, and  
if I pay you \$50, then you fix my computer,

which, in turn, is logically equivalent to (see Example 1.3.15)

You fix my computer if and only if I pay you \$50.

In ordinary discourse, the intended meaning of statements involving logical operators can often (but, not always!) be inferred. However, in mathematics and science, precision is required. Only by carefully defining what we mean by terms such as “if” and “if and only if” can we obtain unambiguous and precise statements. In particular, logic carefully distinguishes among conditional, biconditional, converse, and contrapositive propositions.

### 1.3 Problem-Solving Tips

- In formal logic, “if” and “if and only if” are quite different. The conditional proposition  $p \rightarrow q$  (if  $p$  then  $q$ ) is true except when  $p$  is true and  $q$  is false. On the other hand, the biconditional proposition  $p \leftrightarrow q$  ( $p$  if and only if  $q$ ) is true precisely when  $p$  and  $q$  are both true or both false.
- To determine whether propositions  $P$  and  $Q$ , made up of the propositions  $p_1, \dots, p_n$ , are logically equivalent, write the truth tables for  $P$  and  $Q$ . If all of the entries for  $P$  and  $Q$  are always both true or both false, then  $P$  and  $Q$  are equivalent. If some entry is true for one of  $P$  or  $Q$  and false for the other, then  $P$  and  $Q$  are not equivalent.
- De Morgan’s laws for logic

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

give formulas for negating “or” ( $\vee$ ) and negating “and” ( $\wedge$ ). Roughly speaking, negating “or” results in “and,” and negating “and” results in “or.”

- Example 1.3.13 states a very important equivalence

$$\neg(p \rightarrow q) \equiv p \wedge \neg q,$$

which we will meet throughout this book. This equivalence shows that the negation of the conditional proposition can be written using the “and” ( $\wedge$ ) operator. Notice that there is no conditional operator on the right-hand side of the equation.

## 1.3 Review Exercises

1. What is a conditional proposition? How is it denoted?
2. Give the truth table for the conditional proposition.
3. In a conditional proposition, what is the hypothesis?
4. In a conditional proposition, what is the conclusion?
5. What is a necessary condition?
6. What is a sufficient condition?
7. What is the converse of  $p \rightarrow q$ ?
8. What is a biconditional proposition? How is it denoted?
9. Give the truth table for the biconditional proposition.
10. What does it mean for  $P$  to be logically equivalent to  $Q$ ?
11. State De Morgan’s laws for logic.
12. What is the contrapositive of  $p \rightarrow q$ ?

## 1.3 Exercises

In Exercises 1–11, restate each proposition in the form (1.3.2) of a conditional proposition.

1. Joey will pass the discrete mathematics exam if he studies hard.
2. Rosa may graduate if she has 160 quarter-hours of credits.
3. A necessary condition for Fernando to buy a computer is that he obtain \$2000.
4. A sufficient condition for Katrina to take the algorithms course is that she pass discrete mathematics.
5. Getting that job requires knowing someone who knows the boss.
6. You can go to the Super Bowl unless you can’t afford the ticket.
7. You may inspect the aircraft only if you have the proper security clearance.
8. When better cars are built, Buick will build them.
9. The audience will go to sleep if the chairperson gives the lecture.
10. The program is readable only if it is well structured.
11. A necessary condition for the switch to not be turned properly is that the light is not on.
12. Write the converse of each proposition in Exercises 1–11.
13. Write the contrapositive of each proposition in Exercises 1–11.

Assuming that  $p$  and  $r$  are false and that  $q$  and  $s$  are true, find the truth value of each proposition in Exercises 14–22.

14.  $p \rightarrow q$
15.  $\neg p \rightarrow \neg q$
16.  $\neg(p \rightarrow q)$
17.  $(p \rightarrow q) \wedge (q \rightarrow r)$

18.  $(p \rightarrow q) \rightarrow r$
19.  $p \rightarrow (q \rightarrow r)$
20.  $(s \rightarrow (p \wedge \neg r)) \wedge ((p \rightarrow (r \vee q)) \wedge s)$
21.  $((p \wedge \neg q) \rightarrow (q \wedge r)) \rightarrow (s \vee \neg q)$
22.  $((p \vee q) \wedge (q \vee s)) \rightarrow ((\neg r \vee p) \wedge (q \vee s))$

Exercises 23–32 refer to the propositions  $p$ ,  $q$ , and  $r$ ;  $p$  is true,  $q$  is false, and  $r$ ’s status is unknown at this time. Tell whether each proposition is true, is false, or has unknown status at this time.

23.  $p \vee r$
24.  $p \wedge r$
25.  $p \rightarrow r$
26.  $q \rightarrow r$
27.  $r \rightarrow p$
28.  $r \rightarrow q$
29.  $(p \wedge r) \leftrightarrow r$
30.  $(p \vee r) \leftrightarrow r$
31.  $(q \wedge r) \leftrightarrow r$
32.  $(q \vee r) \leftrightarrow r$

Determine the truth value of each proposition in Exercises 33–42.

33. If  $3 + 5 < 2$ , then  $1 + 3 = 4$ .
34. If  $3 + 5 < 2$ , then  $1 + 3 \neq 4$ .
35. If  $3 + 5 > 2$ , then  $1 + 3 = 4$ .
36. If  $3 + 5 > 2$ , then  $1 + 3 \neq 4$ .
37.  $3 + 5 > 2$  if and only if  $1 + 3 = 4$ .
38.  $3 + 5 < 2$  if and only if  $1 + 3 = 4$ .
39.  $3 + 5 < 2$  if and only if  $1 + 3 \neq 4$ .
40. If the earth has six moons, then  $1 < 3$ .
41. If  $1 < 3$ , then the earth has six moons.
42. If  $\pi < 3.14$ , then  $\pi^2 < 9.85$ .

## 30 Chapter 1 ◆ Sets and Logic

In Exercises 43–46, represent the given proposition symbolically by letting

$$p: 4 < 2, \quad q: 7 < 10, \quad r: 6 < 6.$$

**43.** If  $4 < 2$ , then  $7 < 10$ .

**44.** If  $(4 < 2)$  and  $6 < 6$ , then  $7 < 10$ .

**45.** If it is not the case that  $(6 < 6)$  and  $7$  is not less than  $10$ , then  $6 < 6$ .

**46.**  $7 < 10$  if and only if  $(4 < 2)$  and  $6$  is not less than  $6$ .

In Exercises 47–52, represent the given proposition symbolically by letting

$$p: \text{You run 10 laps daily.}$$

$$q: \text{You are healthy.}$$

$$r: \text{You take multi-vitamins.}$$

**47.** If you run 10 laps daily, then you will be healthy.

**48.** If you do not run 10 laps daily or do not take multi-vitamins, then you will not be healthy.

**49.** Taking multi-vitamins is sufficient for being healthy.

**50.** You will be healthy if and only if you run 10 laps daily and take multi-vitamins.

**51.** If you are healthy, then you run 10 laps daily or you take multi-vitamins.

**52.** If you are healthy and run 10 laps daily, then you do not take multi-vitamins.

In Exercises 53–58, formulate the symbolic expression in words using

$$p: \text{Today is Monday,}$$

$$q: \text{It is raining,}$$

$$r: \text{It is hot.}$$

**53.**  $p \rightarrow q$

**54.**  $\neg q \rightarrow (r \wedge p)$

**55.**  $\neg p \rightarrow (q \vee r)$

**56.**  $\neg(p \vee q) \leftrightarrow r$

**57.**  $(p \wedge (q \vee r)) \rightarrow (r \vee (q \vee p))$

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

In Exercises 59–62, write each conditional proposition symbolically. Write the converse and contrapositive of each proposition symbolically and in words. Also, find the truth value of each conditional proposition, its converse, and its contrapositive.

**59.** If  $4 < 6$ , then  $9 > 12$ .

**60.** If  $4 > 6$ , then  $9 > 12$ .

**61.**  $|1| < 3$  if  $-3 < 1 < 3$ .

**62.**  $|4| < 3$  if  $-3 < 4 < 3$ .

For each pair of propositions  $P$  and  $Q$  in Exercises 63–72, state whether or not  $P \equiv Q$ .

**63.**  $P = p, Q = p \vee q$

**64.**  $P = p \wedge q, Q = \neg p \vee \neg q$

**65.**  $P = p \rightarrow q, Q = \neg p \vee q$

**66.**  $P = p \wedge (\neg q \vee r), Q = p \vee (q \wedge \neg r)$

**67.**  $P = p \wedge (q \vee r), Q = (p \vee q) \wedge (p \vee r)$

**68.**  $P = p \rightarrow q, Q = \neg q \rightarrow \neg p$

**69.**  $P = p \rightarrow q, Q = p \leftrightarrow q$

**70.**  $P = (p \rightarrow q) \wedge (q \rightarrow r), Q = p \rightarrow r$

**71.**  $P = (p \rightarrow q) \rightarrow r, Q = p \rightarrow (q \rightarrow r)$

**72.**  $P = (s \rightarrow (p \wedge \neg r)) \wedge ((p \rightarrow (r \vee q)) \wedge s), Q = p \vee t$

Using De Morgan's laws for logic, write the negation of each proposition in Exercises 73–76.

**73.** Pat will use the treadmill or lift weights.

**74.** Dale is smart and funny.

**75.** Shirley will either take the bus or catch a ride to school.

**76.** Red pepper and onions are required to make chili.

Exercises 77 and 78 provide further motivation for defining  $p \rightarrow q$  to be true when  $p$  is false. We consider changing the truth table for  $p \rightarrow q$  when  $p$  is false. For the first change, we call the resulting operator  $\text{imp1}$  (Exercise 77), and, for the second change, we call the resulting operator  $\text{imp2}$  (Exercise 78). In both cases, we see that pathologies result.

**77.** Define the truth table for  $\text{imp1}$  by

| $p$ | $q$ | $p \text{ imp1 } q$ |
|-----|-----|---------------------|
| T   | T   | T                   |
| T   | F   | F                   |
| F   | T   | F                   |
| F   | F   | T                   |

Show that  $p \text{ imp1 } q \equiv q \text{ imp1 } p$ .

**78.** Define the truth table for  $\text{imp2}$  by

| $p$ | $q$ | $p \text{ imp2 } q$ |
|-----|-----|---------------------|
| T   | T   | T                   |
| T   | F   | F                   |
| F   | T   | T                   |
| F   | F   | F                   |

(a) Show that

$$(p \text{ imp2 } q) \wedge (q \text{ imp2 } p) \not\equiv p \leftrightarrow q. \quad (1.3.6)$$

(b) Show that (1.3.6) remains true if we change the third row of  $\text{imp2}$ 's truth table to F T F.

**79.** Verify the second of De Morgan's laws,  $\neg(p \wedge q) \equiv \neg p \vee \neg q$ .

**80.** Show that  $(p \rightarrow q) \equiv (\neg p \vee q)$ .

## 1.4 Arguments and Rules of Inference

Consider the following sequence of propositions.

- The bug is either in module 17 or in module 81.
- The bug is a numerical error.
- Module 81 has no numerical error. (1.4.1)

Assuming that these statements are true, it is reasonable to conclude

$$\text{The bug is in module 17.} \quad (1.4.2)$$

This process of drawing a conclusion from a sequence of propositions is called **deductive reasoning**. The given propositions, such as (1.4.1), are called **hypotheses** or **premises**, and the proposition that follows from the hypotheses, such as (1.4.2), is called the **conclusion**. A **(deductive) argument** consists of hypotheses together with a conclusion. Many proofs in mathematics and computer science are deductive arguments.

Any argument has the form

$$\text{If } p_1 \text{ and } p_2 \text{ and } \dots \text{ and } p_n, \text{ then } q. \quad (1.4.3)$$

Argument (1.4.3) is said to be **valid** if the conclusion follows from the hypotheses; that is, if  $p_1$  and  $p_2$  and  $\dots$  and  $p_n$  are true, then  $q$  must also be true. This discussion motivates the following definition.

**Definition 1.4.1** ► An *argument* is a sequence of propositions written

$$\begin{array}{c} p_1 \\ p_2 \\ \vdots \\ p_n \\ \hline \therefore q \end{array}$$

or

$$p_1, p_2, \dots, p_n / \therefore q.$$

The symbol  $\therefore$  is read “therefore.” The propositions  $p_1, p_2, \dots, p_n$  are called the *hypotheses* (or *premises*), and the proposition  $q$  is called the *conclusion*. The argument is *valid* provided that if  $p_1$  and  $p_2$  and  $\dots$  and  $p_n$  are all true, then  $q$  must also be true; otherwise, the argument is *invalid* (or a *fallacy*). ◀

### Go Online

For more on fallacies, see [goo.gl/F7b35e](http://goo.gl/F7b35e)

In a valid argument, we sometimes say that the conclusion follows from the hypotheses. Notice that we are not saying that the conclusion is true; we are only saying that if you grant the hypotheses, you must also grant the conclusion. An argument is valid because of its form, not because of its content.

Each step of an extended argument involves drawing intermediate conclusions. For the argument as a whole to be valid, each step of the argument must result in a valid, intermediate conclusion. **Rules of inference**, brief, valid arguments, are used within a larger argument.

**Example 1.4.2** Determine whether the argument

$$p \rightarrow q$$

$$\begin{array}{c} p \\ \hline \end{array}$$

$$\therefore q$$

is valid.

**FIRST SOLUTION** We construct a truth table for all the propositions involved:

| $p$ | $q$ | $p \rightarrow q$ | $p$ | $q$ |
|-----|-----|-------------------|-----|-----|
| T   | T   | T                 | T   | T   |
| T   | F   | F                 | T   | F   |
| F   | T   | T                 | F   | T   |
| F   | F   | T                 | F   | F   |

We observe that whenever the hypotheses  $p \rightarrow q$  and  $p$  are true, the conclusion  $q$  is also true; therefore, the argument is valid.

**SECOND SOLUTION** We can avoid writing the truth table by directly verifying that whenever the hypotheses are true, the conclusion is also true.

Suppose that  $p \rightarrow q$  and  $p$  are true. Then  $q$  must be true, for otherwise  $p \rightarrow q$  would be false. Therefore, the argument is valid. 

The argument in Example 1.4.2 is used extensively and is known as the **modus ponens rule of inference** or **law of detachment**. Several useful rules of inference for propositions, which may be verified using truth tables (see Exercises 33–38), are listed in Table 1.4.1.

**TABLE 1.4.1** ■ Rules of Inference for Propositions

| Rule of Inference                                          | Name           | Rule of Inference                                                            | Name                      |
|------------------------------------------------------------|----------------|------------------------------------------------------------------------------|---------------------------|
| $\frac{p \rightarrow q}{\therefore q}$                     | Modus ponens   | $\frac{p}{\frac{q}{\therefore p \wedge q}}$                                  | Conjunction               |
| $\frac{p \rightarrow q}{\frac{\neg q}{\therefore \neg p}}$ | Modus tollens  | $\frac{p \rightarrow q}{\frac{q \rightarrow r}{\therefore p \rightarrow r}}$ | Hypothetical<br>syllogism |
| $\frac{p}{\therefore p \vee q}$                            | Addition       | $\frac{p \vee q}{\frac{\neg p}{\therefore q}}$                               | Disjunctive<br>syllogism  |
| $\frac{p \wedge q}{\therefore p}$                          | Simplification |                                                                              |                           |

**Example 1.4.3**

Which rule of inference is used in the following argument?

If the computer has one gigabyte of memory, then it can run “Blast ‘em.” If the computer can run “Blast ‘em,” then the sonics will be impressive. Therefore, if the computer has one gigabyte of memory, then the sonics will be impressive.

**SOLUTION** Let  $p$  denote the proposition “the computer has one gigabyte of memory,” let  $q$  denote the proposition “the computer can run ‘Blast ’em,’” and let  $r$  denote the proposition “the sonics will be impressive.” The argument can be written symbolically as

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

Therefore, the argument uses the hypothetical syllogism rule of inference. ▲

#### Example 1.4.4

Represent the argument

$$\begin{array}{c} \text{If } 2 = 3, \text{ then I ate my hat.} \\ \text{I ate my hat.} \\ \hline \therefore 2 = 3 \end{array}$$

symbolically and determine whether the argument is valid.

**SOLUTION** If we let  $p$ :  $2 = 3$  and  $q$ : I ate my hat, the argument may be written

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

If the argument is valid, then whenever  $p \rightarrow q$  and  $q$  are both true,  $p$  must also be true. Suppose that  $p \rightarrow q$  and  $q$  are true. This is possible if  $p$  is false and  $q$  is true. In this case,  $p$  is not true; thus the argument is invalid. This fallacy is known as the **fallacy of affirming the conclusion**. ▲

We can also determine whether the argument in Example 1.4.4 is valid or not by examining the truth table of Example 1.4.2. In the third row of the table, the hypotheses are true and the conclusion is false; thus the argument is invalid.

#### Example 1.4.5

Represent the argument

$$\begin{array}{c} \text{The bug is either in module 17 or in module 81.} \\ \text{The bug is a numerical error.} \\ \text{Module 81 has no numerical error.} \\ \hline \therefore \text{The bug is in module 17.} \end{array}$$

given at the beginning of this section symbolically and show that it is valid.

**SOLUTION** If we let

$$\begin{array}{l} p : \text{The bug is in module 17.} \\ q : \text{The bug is in module 81.} \\ r : \text{The bug is a numerical error.} \end{array}$$

the argument may be written

$$\begin{array}{c} p \vee q \\ r \\ r \rightarrow \neg q \\ \hline \therefore p \end{array}$$

From  $r \rightarrow \neg q$  and  $r$ , we may use modus ponens to conclude  $\neg q$ . From  $p \vee q$  and  $\neg q$ , we may use the disjunctive syllogism to conclude  $p$ . Thus the conclusion  $p$  follows from the hypotheses and the argument is valid. 

### Example 1.4.6

We are given the following hypotheses: If the Chargers get a good linebacker, then the Chargers can beat the Broncos. If the Chargers can beat the Broncos, then the Chargers can beat the Jets. If the Chargers can beat the Broncos, then the Chargers can beat the Dolphins. The Chargers get a good linebacker. Show by using the rules of inference (see Table 1.4.1) that the conclusion, the Chargers can beat the Jets and the Chargers can beat the Dolphins, follows from the hypotheses.

**SOLUTION** Let  $p$  denote the proposition “the Chargers get a good linebacker,” let  $q$  denote the proposition “the Chargers can beat the Broncos,” let  $r$  denote the proposition “the Chargers can beat the Jets,” and let  $s$  denote the proposition “the Chargers can beat the Dolphins.” Then the hypotheses are:

$$\begin{array}{l} p \rightarrow q \\ q \rightarrow r \\ q \rightarrow s \\ p. \end{array}$$

From  $p \rightarrow q$  and  $q \rightarrow r$ , we may use the hypothetical syllogism to conclude  $p \rightarrow r$ . From  $p \rightarrow r$  and  $p$ , we may use modus ponens to conclude  $r$ . From  $p \rightarrow q$  and  $q \rightarrow s$ , we may use the hypothetical syllogism to conclude  $p \rightarrow s$ . From  $p \rightarrow s$  and  $p$ , we may use modus ponens to conclude  $s$ . From  $r$  and  $s$ , we may use conjunction to conclude  $r \wedge s$ . Since  $r \wedge s$  represents the proposition “the Chargers can beat the Jets and the Chargers can beat the Dolphins,” we conclude that the conclusion does follow from the hypotheses. 

### 1.4 Problem-Solving Tips

The validity of a very short argument or proof might be verified using a truth table. In practice, arguments and proofs use rules of inference.

## 1.4 Review Exercises

1. What is deductive reasoning?
2. What is a hypothesis in an argument?
3. What is a premise in an argument?
4. What is a conclusion in an argument?
5. What is a valid argument?
6. What is an invalid argument?
7. State the modus ponens rule of inference.
8. State the modus tollens rule of inference.
9. State the addition rule of inference.
10. State the simplification rule of inference.
11. State the conjunction rule of inference.
12. State the hypothetical syllogism rule of inference.
13. State the disjunctive syllogism rule of inference.

## 1.4 Exercises

Formulate the arguments of Exercises 1–5 symbolically and determine whether each is valid. Let

$$p: I \text{ study hard.} \quad q: I \text{ get A's.} \quad r: I \text{ get rich.}$$

1. If I study hard, then I get A's.

I study hard.

---

∴ I get A's.

2. If I study hard, then I get A's.

If I don't get rich, then I don't get A's.

---

∴ I get rich.

3. I study hard if and only if I get rich.

I get rich.

---

∴ I study hard.

4. If I study hard or I get rich, then I get A's.

I get A's.

---

∴ If I don't study hard, then I get rich.

5. If I study hard, then I get A's or I get rich.

I don't get A's and I don't get rich.

---

∴ I don't study hard.

Formulate the arguments of Exercises 6–9 symbolically and determine whether each is valid.

*p: The Democrats win.*

*q: The Republicans win.*

*r: Unemployment is up.*

*s: The economy is up.*

6. If the Democrats win, then the economy is up; and, if the Republicans win, then unemployment is up.

The Democrats win or the Republicans win.

---

∴ Unemployment is up or the economy is up.

7. If the Democrats win or the Republicans win, then unemployment is up or the economy is up.

The Democrats win and unemployment is not up.

---

∴ The economy is up.

8. If the Democrats win, then unemployment is up.

If the Republicans win, then the economy is up

The Republicans and Democrats do not both win.

The Democrats do not win.

---

∴ The economy is up.

9. If the Democrats win, then unemployment is up or the economy is up.

If the Republicans win, then unemployment is up.

The economy is not up.

The Democrats win

---

∴ Unemployment is up or the Republicans win.

In Exercises 10–14, write the given argument in words and determine whether each argument is valid. Let

*p: 4 gigabytes is better than no memory at all.*

*q: We will buy more memory.*

*r: We will buy a new computer.*

$$\begin{array}{c} 10. \frac{p \rightarrow r \quad p \rightarrow q}{\therefore p \rightarrow (r \wedge q)} \quad 11. \frac{p \rightarrow (r \vee q) \quad r \rightarrow \neg q}{\therefore p \rightarrow r} \quad 12. \frac{p \rightarrow r \quad r \rightarrow q}{\therefore q} \end{array}$$

$$\begin{array}{c} 13. \frac{\neg r \rightarrow \neg p \quad r}{\therefore p} \quad 14. \frac{p \rightarrow r \quad r \rightarrow q \quad p}{\therefore q} \end{array}$$

In Exercises 15–19, write the given argument in words and determine whether each argument is valid. Let

*p: The for loop is faulty.*

*q: The while loop is faulty.*

*r: The hardware is unreliable.*

*s: The output is correct.*

$$\begin{array}{c} 15. \frac{(p \vee q) \rightarrow (r \vee s) \quad p \quad \neg r}{\therefore s} \quad 16. \frac{(r \vee s) \rightarrow p \quad s \rightarrow q \quad p \vee s}{\therefore r} \end{array}$$

$$\begin{array}{c} 17. \frac{(p \rightarrow r) \rightarrow q \quad (q \rightarrow s) \rightarrow p \quad r \wedge s \quad p \vee q}{\therefore p \wedge q} \quad 18. \frac{p \rightarrow q \quad q \rightarrow r \quad \neg r \quad s \rightarrow r}{\therefore \neg s} \end{array}$$

$$\begin{array}{c} 19. \frac{p \rightarrow (q \vee r) \quad q \rightarrow (p \vee s) \quad p \vee \neg q \quad \neg s}{\therefore p \vee r} \end{array}$$

Determine whether each argument in Exercises 20–24 is valid.

$$\begin{array}{c} 20. \frac{p \rightarrow q \quad \neg p}{\therefore \neg q} \quad 21. \frac{p \rightarrow q \quad \neg q}{\therefore \neg p} \quad 22. \frac{p \wedge \neg p}{\therefore q} \end{array}$$

23. 
$$\begin{array}{c} p \rightarrow (q \rightarrow r) \\ q \rightarrow (p \rightarrow r) \\ \hline \therefore (p \vee q) \rightarrow r \end{array}$$

24. 
$$\begin{array}{c} (p \rightarrow q) \wedge (r \rightarrow s) \\ p \vee r \\ \hline \therefore q \vee s \end{array}$$

25. Show that if

$$p_1, p_2 / \therefore p \quad \text{and} \quad p, p_3, \dots, p_n / \therefore c$$

are valid arguments, the argument

$$p_1, p_2, \dots, p_n / \therefore c$$

is also valid.

26. Comment on the following argument:

Hard disk drive storage is better than nothing.

Nothing is better than a solid state drive.

$\therefore$  Hard disk drive is better than a solid state drive.

For each argument in Exercises 27–29, tell which rule of inference is used.

27. Fishing is a popular sport. Therefore, fishing is a popular sport or lacrosse is wildly popular in California.  
 28. If fishing is a popular sport, then lacrosse is wildly popular in California. Fishing is a popular sport. Therefore, lacrosse is wildly popular in California.

29. Fishing is a popular sport or lacrosse is wildly popular in California. Lacrosse is not wildly popular in California. Therefore, fishing is a popular sport.

In Exercises 30–32, give an argument using rules of inference to show that the conclusion follows from the hypotheses.

30. Hypotheses: If there is gas in the car, then I will go to the store. If I go to the store, then I will get a soda. There is gas in the car. Conclusion: I will get a soda.  
 31. Hypotheses: If there is gas in the car, then I will go to the store. If I go to the store, then I will get a soda. I do not get a soda. Conclusion: There is not gas in the car, or the car transmission is defective.  
 32. Hypotheses: If Jill can sing or Dweezle can play, then I'll buy the compact disc. Jill can sing. I'll buy the compact disc player. Conclusion: I'll buy the compact disc and the compact disc player.  
 33. Show that modus tollens (see Table 1.4.1) is valid.  
 34. Show that addition (see Table 1.4.1) is valid.  
 35. Show that simplification (see Table 1.4.1) is valid.  
 36. Show that conjunction (see Table 1.4.1) is valid.  
 37. Show that hypothetical syllogism (see Table 1.4.1) is valid.  
 38. Show that disjunctive syllogism (see Table 1.4.1) is valid.

## 1.5 Quantifiers

### Go Online

For more on quantifiers, see [goo.gl/F7b35e](http://goo.gl/F7b35e)

The logic in Sections 1.2 and 1.3 that deals with propositions is incapable of describing most of the statements in mathematics and computer science. Consider, for example, the statement

$$p: n \text{ is an odd integer.}$$

A proposition is a statement that is either true or false. The statement  $p$  is not a proposition, because whether  $p$  is true or false depends on the value of  $n$ . For example,  $p$  is true if  $n = 103$  and false if  $n = 8$ . Since most of the statements in mathematics and computer science use variables, we must extend the system of logic to include such statements.

**Definition 1.5.1** ► Let  $P(x)$  be a statement involving the variable  $x$  and let  $D$  be a set. We call  $P$  a *propositional function* or *predicate* (with respect to  $D$ ) if for each  $x \in D$ ,  $P(x)$  is a proposition. We call  $D$  the *domain of discourse* of  $P$ .

In Definition 1.5.1, the domain of discourse specifies the allowable values for  $x$ .

### Example 1.5.2

Let  $P(n)$  be the statement

$$n \text{ is an odd integer.}$$

Then  $P$  is a propositional function with domain of discourse  $\mathbf{Z}^+$ , since for each  $n \in \mathbf{Z}^+$ ,  $P(n)$  is a proposition [i.e., for each  $n \in \mathbf{Z}^+$ ,  $P(n)$  is true or false but not both]. For example, if  $n = 1$ , we obtain the proposition

$$P(1): 1 \text{ is an odd integer}$$

(which is true). If  $n = 2$ , we obtain the proposition

$$P(2): 2 \text{ is an odd integer}$$

(which is false). 

A propositional function  $P$ , by itself, is neither true nor false. However, for each  $x$  in the domain of discourse,  $P(x)$  is a proposition and is, therefore, either true or false. We can think of a propositional function as defining a class of propositions, one for each element in the domain of discourse. For example, if  $P$  is a propositional function with domain of discourse  $\mathbf{Z}^+$ , we obtain the class of propositions

$$P(1), P(2), \dots$$

Each of  $P(1), P(2), \dots$  is either true or false.

### Example 1.5.3

Explain why the following are propositional functions.

- (a)  $n^2 + 2n$  is an odd integer (domain of discourse =  $\mathbf{Z}^+$ ).
- (b)  $x^2 - x - 6 = 0$  (domain of discourse =  $\mathbf{R}$ ).
- (c) The baseball player hit over .300 in 2015 (domain of discourse = set of baseball players).
- (d) The film is rated over 20% by Rotten Tomatoes [the scale is 0% (awful) to 100% (terrific)]. The domain of discourse is the set of films rated by Rotten Tomatoes.

**SOLUTION** In statement (a), for each positive integer  $n$ , we obtain a proposition; therefore, statement (a) is a propositional function.

Similarly, in statement (b), for each real number  $x$ , we obtain a proposition; therefore, statement (b) is a propositional function.

We can regard the variable in statement (c) as “baseball player.” Whenever we substitute a particular baseball player for the variable “baseball player,” the statement is a proposition. For example, if we substitute “Joey Votto” for “baseball player,” statement (c) is

Joey Votto hit over .300 in 2015,

which is true. If we substitute “Andrew McCutchen” for “baseball player,” statement (c) is

Andrew McCutchen hit over .300 in 2015,

which is false. Thus statement (c) is a propositional function.

Statement (d) is similar in form to statement (c). Here the variable is “film.” Whenever we substitute a film rated by Rotten Tomatoes for the variable “film,” the statement is a proposition. For example if we substitute *Spectre* for “film,” statement (d) is

*Spectre* is rated over 20% by Rotten Tomatoes,

which is true since *Spectre* is rated 64% by Rotten Tomatoes. If we substitute *Blended* for “film,” statement (d) is

*Blended* is rated over 20% by Rotten Tomatoes,

which is false since *Blended* is rated 14% by Rotten Tomatoes. Thus statement (d) is a propositional function. ▲

Most of the statements in mathematics and computer science use terms such as “for every” and “for some.” For example, in mathematics we have the following theorem:

For every triangle  $T$ , the sum of the angles of  $T$  is equal to  $180^\circ$ .

In computer science, we have this theorem:

For some program  $P$ , the output of  $P$  is  $P$  itself.

We now extend the logical system of Sections 1.2 and 1.3 so that we can handle statements that include “for every” and “for some.”

**Definition 1.5.4** ► Let  $P$  be a propositional function with domain of discourse  $D$ . The statement

$$\text{for every } x, P(x)$$

is said to be a *universally quantified statement*. The symbol  $\forall$  means “for every.” Thus the statement

$$\text{for every } x, P(x)$$

may be written

$$\forall x P(x). \quad (1.5.1)$$

The symbol  $\forall$  is called a *universal quantifier*.

The statement (1.5.1) is true if  $P(x)$  is true for every  $x$  in  $D$ . The statement (1.5.1) is false if  $P(x)$  is false for at least one  $x$  in  $D$ . ▲

### Example 1.5.5

Consider the universally quantified statement

$$\forall x(x^2 \geq 0).$$

The domain of discourse is  $\mathbf{R}$ . The statement is true because, *for every* real number  $x$ , it is true that the square of  $x$  is positive or zero. ▲

According to Definition 1.5.4, the universally quantified statement (1.5.1) is false if *for at least one*  $x$  in the domain of discourse, the proposition  $P(x)$  is false. A value  $x$  in the domain of discourse that makes  $P(x)$  false is called a **counterexample** to the statement (1.5.1).

### Example 1.5.6

Determine whether the universally quantified statement  $\forall x(x^2 - 1 > 0)$  is true or false. The domain of discourse is  $\mathbf{R}$ .

**SOLUTION** The statement is false since, if  $x = 1$ , the proposition  $1^2 - 1 > 0$  is false. The value 1 is a counterexample to the statement  $\forall x(x^2 - 1 > 0)$ . Although there are values of  $x$  that make the propositional function true, the counterexample provided shows that the universally quantified statement is false. ▲

### Example 1.5.7

Suppose that  $P$  is a propositional function whose domain of discourse is the set  $\{d_1, \dots, d_n\}$ . The following pseudocode<sup>†</sup> determines whether  $\forall x P(x)$  is true or false:

```
for i = 1 to n
 if ($\neg P(d_i)$)
 return false
 return true
```

The for loop examines the members  $d_i$  of the domain of discourse one by one. If it finds a value  $d_i$  for which  $P(d_i)$  is false, the condition  $\neg P(d_i)$  in the if statement is true; so the code returns false [to indicate that  $\forall x P(x)$  is false] and terminates. In this case,  $d_i$  is a counterexample. If  $P(d_i)$  is true for every  $d_i$ , the condition  $\neg P(d_i)$  in the if statement is always false. In this case, the for loop runs to completion, after which the code returns true [to indicate that  $\forall x P(x)$  is true] and terminates.

Notice that if  $\forall x P(x)$  is true, the for loop necessarily runs to completion so that *every* member of the domain of discourse is checked to ensure that  $P(x)$  is true for every  $x$ . If  $\forall x P(x)$  is false, the for loop terminates as soon as *one* element  $x$  of the domain of discourse is found for which  $P(x)$  is false. ▲

We call the variable  $x$  in the propositional function  $P(x)$  a *free variable*. (The idea is that  $x$  is “free” to roam over the domain of discourse.) We call the variable  $x$  in the universally quantified statement  $\forall x P(x)$  a *bound variable*. (The idea is that  $x$  is “bound” by the quantifier  $\forall$ .)

We previously pointed out that a propositional function does not have a truth value. On the other hand, Definition 1.5.4 assigns a truth value to the quantified statement  $\forall x P(x)$ . In sum, a statement with free (unquantified) variables is not a proposition, and a statement with no free variables (no unquantified variables) is a proposition.

Alternative ways to write  $\forall x P(x)$  are

for all  $x$ ,  $P(x)$

and

for any  $x$ ,  $P(x)$ .

The symbol  $\forall$  may be read “for every,” “for all,” or “for any.”

To prove that  $\forall x P(x)$  is *true*, we must, in effect, examine *every* value of  $x$  in the domain of discourse and show that for every  $x$ ,  $P(x)$  is true. One technique for proving that  $\forall x P(x)$  is true is to let  $x$  denote an *arbitrary* element of the domain of discourse  $D$ . The argument then proceeds using the symbol  $x$ . Whatever is claimed about  $x$  must be true *no matter what value*  $x$  might have in  $D$ . The argument must conclude by proving that  $P(x)$  is true.

Sometimes to specify the domain of discourse  $D$ , we write a universally quantified statement as

for every  $x$  in  $D$ ,  $P(x)$ .

---

<sup>†</sup>The pseudocode used in this book is explained in Appendix C.

**Example 1.5.8** Verify that the universally quantified statement

for every real number  $x$ , if  $x > 1$ , then  $x + 1 > 1$

is true.

**SOLUTION** This time we must verify that the statement

$$\text{if } x > 1, \text{ then } x + 1 > 1 \quad (1.5.2)$$

is true *for every* real number  $x$ .

Let  $x$  be any real number whatsoever. It is true that for any real number  $x$ , either  $x \leq 1$  or  $x > 1$ . If  $x \leq 1$ , the conditional proposition (1.5.2) is vacuously true. (The proposition is true because the hypothesis  $x > 1$  is false. Recall that when the hypothesis is false, the conditional proposition is true regardless of whether the conclusion is true or false.) In most arguments, the vacuous case is omitted.

Now suppose that  $x > 1$ . Regardless of the specific value of  $x$ ,  $x + 1 > x$ . Since  $x + 1 > x$  and  $x > 1$ , we conclude that  $x + 1 > 1$ , so the conclusion is true. If  $x > 1$ , the hypothesis and conclusion are both true; hence the conditional proposition (1.5.2) is true.

We have shown that for every real number  $x$ , the proposition (1.5.2) is true. Therefore, the universally quantified statement

$$\text{for every real number } x, \text{ if } x > 1, \text{ then } x + 1 > 1$$

is true. ◀

The method of disproving the statement  $\forall x P(x)$  is quite different from the method used to prove that the statement is true. To show that the universally quantified statement  $\forall x P(x)$  is *false*, it is sufficient to find *one* value  $x$  in the domain of discourse for which the proposition  $P(x)$  is false. Such a value, we recall, is called a counterexample to the universally quantified statement.

We turn next to existentially quantified statements.

**Definition 1.5.9** ▶ Let  $P$  be a propositional function with domain of discourse  $D$ . The statement

$$\text{there exists } x, P(x)$$

is said to be an *existentially quantified statement*. The symbol  $\exists$  means “there exists.” Thus the statement

$$\text{there exists } x, P(x)$$

may be written

$$\exists x P(x). \quad (1.5.3)$$

The symbol  $\exists$  is called an *existential quantifier*.

The statement (1.5.3) is true if  $P(x)$  is true for at least one  $x$  in  $D$ . The statement (1.5.3) is false if  $P(x)$  is false for every  $x$  in  $D$ . ◀

**Example 1.5.10** Consider the existentially quantified statement

$$\exists x \left( \frac{x}{x^2 + 1} = \frac{2}{5} \right).$$

The domain of discourse is  $\mathbf{R}$ . The statement is true because it is possible to find *at least one* real number  $x$  for which the proposition

$$\frac{x}{x^2 + 1} = \frac{2}{5}$$

is true. For example, if  $x = 2$ , we obtain the true proposition

$$\frac{2}{2^2 + 1} = \frac{2}{5}.$$

It is not the case that *every* value of  $x$  results in a true proposition. For example, if  $x = 1$ , the proposition

$$\frac{1}{1^2 + 1} = \frac{2}{5}$$

is false. 

According to Definition 1.5.9, the existentially quantified statement (1.5.3) is false if for every  $x$  in the domain of discourse, the proposition  $P(x)$  is false.

**Example 1.5.11** Verify that the existentially quantified statement

$$\exists x \in \mathbf{R} \left( \frac{1}{x^2 + 1} > 1 \right)$$

is false.

**SOLUTION** We must show that

$$\frac{1}{x^2 + 1} > 1$$

is false for every real number  $x$ . Now

$$\frac{1}{x^2 + 1} > 1$$

is false precisely when

$$\frac{1}{x^2 + 1} \leq 1 \tag{1.5.4}$$

is true. Thus, we must show that (1.5.4) is true for every real number  $x$ . To this end, let  $x$  be any real number whatsoever. Since  $0 \leq x^2$ , we may add 1 to both sides of this inequality to obtain  $1 \leq x^2 + 1$ . If we divide both sides of this last inequality by  $x^2 + 1$ , we obtain (1.5.4). Therefore, the statement (1.5.4) is true for every real number  $x$ . Thus the statement

$$\frac{1}{x^2 + 1} > 1$$

is false for every real number  $x$ . We have shown that the existentially quantified statement

$$\exists x \left( \frac{1}{x^2 + 1} > 1 \right)$$

is false. 

**Example 1.5.12** Suppose that  $P$  is a propositional function whose domain of discourse is the set  $\{d_1, \dots, d_n\}$ . The following pseudocode determines whether  $\exists x P(x)$  is true or false:

```
for i = 1 to n
 if (P(di))
 return true
 return false
```

The for loop examines the members  $d_i$  in the domain of discourse one by one. If it finds a value  $d_i$  for which  $P(d_i)$  is true, the condition  $P(d_i)$  in the if statement is true; so the code returns true [to indicate that  $\exists x P(x)$  is true] and terminates. In this case, the code found a value in the domain of discourse, namely  $d_i$ , for which  $P(d_i)$  is true. If  $P(d_i)$  is false for every  $d_i$ , the condition  $P(d_i)$  in the if statement is always false. In this case, the for loop runs to completion, after which the code returns false [to indicate that  $\exists x P(x)$  is true] and terminates.

Notice that if  $\exists x P(x)$  is true, the for loop terminates as soon as *one* element  $x$  in the domain of discourse is found for which  $P(x)$  is true. If  $\exists x P(x)$  is false, the for loop necessarily runs to completion so that *every* member in the domain of discourse is checked to ensure that  $P(x)$  is false for every  $x$ . 

Alternative ways to write  $\exists x P(x)$  are

there exists  $x$  such that,  $P(x)$

and

for some  $x$ ,  $P(x)$

and

for at least one  $x$ ,  $P(x)$ .

The symbol  $\exists$  may be read “there exists,” “for some,” or “for at least one.”

**Example 1.5.13** Consider the existentially quantified statement

for some  $n$ , if  $n$  is prime, then  $n + 1, n + 2, n + 3$ , and  $n + 4$  are not prime.

The domain of discourse is  $\mathbb{Z}^+$ . This statement is true because we can find *at least one* positive integer  $n$  that makes the conditional proposition

if  $n$  is prime, then  $n + 1, n + 2, n + 3$ , and  $n + 4$  are not prime

true. For example, if  $n = 23$ , we obtain the true proposition

if 23 is prime, then 24, 25, 26, and 27 are not prime.

(This conditional proposition is true because both the hypothesis “23 is prime” and the conclusion “24, 25, 26, and 27 are not prime” are true.) Some values of  $n$  make the conditional proposition true (e.g.,  $n = 23, n = 4, n = 47$ ), while others make it false (e.g.,  $n = 2, n = 101$ ). The point is that we found *one* value that makes the conditional proposition

if  $n$  is prime, then  $n + 1, n + 2, n + 3$ , and  $n + 4$  are not prime

true. For this reason, the existentially quantified statement

for some  $n$ , if  $n$  is prime, then  $n + 1, n + 2, n + 3$ , and  $n + 4$  are not prime

is true. 

In Example 1.5.11, we showed that an existentially quantified statement was false by proving that a related universally quantified statement was true. The following theorem makes this relationship precise. The theorem generalizes De Morgan's laws of logic (Example 1.3.11).

### Theorem 1.5.14

#### Generalized De Morgan's Laws for Logic

If  $P$  is a propositional function, each pair of propositions in (a) and (b) has the same truth values (i.e., either both are true or both are false).

- (a)  $\neg(\forall x P(x))$ ;  $\exists x \neg P(x)$
- (b)  $\neg(\exists x P(x))$ ;  $\forall x \neg P(x)$

**Proof** We prove only part (a) and leave the proof of part (b) to the reader (Exercise 73).

Suppose that the proposition  $\neg(\forall x P(x))$  is true. Then the proposition  $\forall x P(x)$  is false. By Definition 1.5.4, the proposition  $\forall x P(x)$  is false precisely when  $P(x)$  is false for at least one  $x$  in the domain of discourse. But if  $P(x)$  is false for at least one  $x$  in the domain of discourse,  $\neg P(x)$  is true for at least one  $x$  in the domain of discourse. By Definition 1.5.9, when  $\neg P(x)$  is true for at least one  $x$  in the domain of discourse, the proposition  $\exists x \neg P(x)$  is true. Thus, if the proposition  $\neg(\forall x P(x))$  is true, the proposition  $\exists x \neg P(x)$  is true. Similarly, if the proposition  $\neg(\forall x P(x))$  is false, the proposition  $\exists x \neg P(x)$  is false.

Therefore, the pair of propositions in part (a) always has the same truth values. 

### Example 1.5.15

Let  $P(x)$  be the statement

$$\frac{1}{x^2 + 1} > 1.$$

In Example 1.5.11 we showed that  $\exists x P(x)$  is false by verifying that

$$\forall x \neg P(x) \tag{1.5.5}$$

is true.

The technique can be justified by appealing to Theorem 1.5.14. After we prove that proposition (1.5.5) is true, we may use Theorem 1.5.14, part (b), to conclude that  $\neg(\exists x P(x))$  is also true. Thus  $\neg\neg(\exists x P(x))$  or, equivalently,  $\exists x P(x)$  is false. 

### Example 1.5.16

Write the statement

Every rock fan loves U2,

symbolically. Write its negation symbolically and in words.

**SOLUTION** Let  $P(x)$  be the propositional function “ $x$  loves U2.” The given statement can be written symbolically as  $\forall x P(x)$ . The domain of discourse is the set of rock fans.

By Theorem 1.5.14, part (a), the negation of the preceding proposition  $\neg(\forall x P(x))$  is equivalent to  $\exists x \neg P(x)$ . In words, this last proposition can be stated as: There exists a rock fan who does not love U2. 

**Example 1.5.17**

Write the statement

Some birds cannot fly,

symbolically. Write its negation symbolically and in words.

**SOLUTION** Let  $P(x)$  be the propositional function “ $x$  flies.” The given statement can be written symbolically as  $\exists x \neg P(x)$ . [The statement could also be written  $\exists x Q(x)$ , where  $Q(x)$  is the propositional function “ $x$  cannot fly.” As in algebra, there are many ways to represent text symbolically.] The domain of discourse is the set of birds.

By Theorem 1.5.14, part (b), the negation  $\neg(\exists x \neg P(x))$  of the preceding proposition is equivalent to  $\forall x \neg \neg P(x)$  or, equivalently,  $\forall x P(x)$ . In words, this last proposition can be stated as: Every bird can fly. 

A universally quantified proposition generalizes the proposition

$$P_1 \wedge P_2 \wedge \cdots \wedge P_n \quad (1.5.6)$$

in the sense that the individual propositions  $P_1, P_2, \dots, P_n$  are replaced by an arbitrary family  $P(x)$ , where  $x$  is in the domain of discourse, and (1.5.6) is replaced by

$$\forall x P(x). \quad (1.5.7)$$

The proposition (1.5.6) is true if and only if  $P_i$  is true for every  $i = 1, \dots, n$ . The truth value of proposition (1.5.7) is defined similarly: (1.5.7) is true if and only if  $P(x)$  is true for every  $x$  in the domain of discourse.

**Example 1.5.18**

Suppose that the domain of discourse of the propositional function  $P$  is  $\{-1, 0, 1\}$ . The propositional function  $\forall x P(x)$  is equivalent to

$$P(-1) \wedge P(0) \wedge P(1). \quad \blacktriangleleft$$

Similarly, an existentially quantified proposition generalizes the proposition

$$P_1 \vee P_2 \vee \cdots \vee P_n \quad (1.5.8)$$

in the sense that the individual propositions  $P_1, P_2, \dots, P_n$  are replaced by an arbitrary family  $P(x)$ , where  $x$  is in the domain of discourse, and (1.5.8) is replaced by  $\exists x P(x)$ .

**Example 1.5.19**

Suppose that the domain of discourse of the propositional function  $P$  is  $\{1, 2, 3, 4\}$ . The propositional function  $\exists x P(x)$  is equivalent to

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

The preceding observations explain how Theorem 1.5.14 generalizes De Morgan’s laws for logic (Example 1.3.11). Recall that the first of De Morgan’s law for logic states that the propositions

$$\neg(P_1 \vee P_2 \vee \cdots \vee P_n) \quad \text{and} \quad \neg P_1 \wedge \neg P_2 \wedge \cdots \wedge \neg P_n$$

have the same truth values. In Theorem 1.5.14, part (b),

$$\neg P_1 \wedge \neg P_2 \wedge \cdots \wedge \neg P_n$$

is replaced by  $\forall x \neg P(x)$  and

$$\neg(P_1 \vee P_2 \vee \cdots \vee P_n)$$

is replaced by  $\neg(\exists x P(x))$ .

### Example 1.5.20

Statements in words often have more than one possible interpretation. Consider the well-known quotation from Shakespeare's "The Merchant of Venice":

All that glitters is not gold.

One possible interpretation of this quotation is: Every object that glitters is not gold. However, this is surely not what Shakespeare intended. The correct interpretation is: Some object that glitters is not gold.

If we let  $P(x)$  be the propositional function " $x$  glitters" and  $Q(x)$  be the propositional function " $x$  is gold," the first interpretation becomes

$$\forall x(P(x) \rightarrow \neg Q(x)), \quad (1.5.9)$$

and the second interpretation becomes

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

Using the result of Example 1.3.13, we see that the truth values of

$$\exists x(P(x) \wedge \neg Q(x))$$

and

$$\exists x \neg(P(x) \rightarrow Q(x))$$

are the same. By Theorem 1.5.14, the truth values of

$$\exists x \neg(P(x) \rightarrow Q(x))$$

and

$$\neg(\forall x P(x) \rightarrow Q(x))$$

are the same. Thus an equivalent way to represent the second interpretation is

$$\neg(\forall x P(x) \rightarrow Q(x)). \quad (1.5.10)$$

Comparing (1.5.9) and (1.5.10), we see that the ambiguity results from whether the negation applies to  $Q(x)$  (the first interpretation) or to the entire statement

$$\forall x(P(x) \rightarrow Q(x))$$

(the second interpretation). The correct interpretation of the statement

All that glitters is not gold

results from negating the entire statement.

In positive statements, "any," "all," "each," and "every" have the same meaning. In negative statements, the situation changes:

Not all  $x$  satisfy  $P(x)$ .

Not each  $x$  satisfies  $P(x)$ .

Not every  $x$  satisfies  $P(x)$ .

are considered to have the same meaning as

For some  $x$ ,  $\neg P(x)$ ;

whereas

Not any  $x$  satisfies  $P(x)$ .

No  $x$  satisfies  $P(x)$ .

mean

For all  $x$ ,  $\neg P(x)$ .

See Exercises 61–71 for other examples. 

### Rules of Inference for Quantified Statements

We conclude this section by introducing some rules of inference for quantified statements and showing how they can be used with rules of inference for propositions (see Section 1.4).

Suppose that  $\forall x P(x)$  is true. By Definition 1.5.4,  $P(x)$  is true for every  $x$  in  $D$ , the domain of discourse. In particular, if  $d$  is in  $D$ , then  $P(d)$  is true. We have shown that the argument

$$\begin{array}{c} \forall x P(x) \\ \hline \therefore P(d) \text{ if } d \in D \end{array}$$

is valid. This rule of inference is called **universal instantiation**. Similar arguments (see Exercises 79–81) justify the other rules of inference listed in Table 1.5.1.

**TABLE 1.5.1 ■ Rules of Inference for Quantified Statements<sup>†</sup>**

| <i>Rule of Inference</i>                                            | <i>Name</i>                |
|---------------------------------------------------------------------|----------------------------|
| $\frac{\forall x P(x)}{\therefore P(d) \text{ if } d \in D}$        | Universal instantiation    |
| $\frac{P(d) \text{ for every } d \in D}{\therefore \forall x P(x)}$ | Universal generalization   |
| $\frac{\exists x P(x)}{\therefore P(d) \text{ for some } d \in D}$  | Existential instantiation  |
| $\frac{P(d) \text{ for some } d \in D}{\therefore \exists x P(x)}$  | Existential generalization |

<sup>†</sup> The domain of discourse is  $D$ .

#### Example 1.5.21

Given that

for every positive integer  $n$ ,  $n^2 \geq n$

is true, we may use universal instantiation to conclude that  $54^2 \geq 54$  since 54 is a positive integer (i.e., a member of the domain of discourse). 

**Example 1.5.22**

Let  $P(x)$  denote the propositional function “ $x$  owns a laptop computer,” where the domain of discourse is the set of students taking MATH 201 (discrete mathematics). Suppose that Taylor, who is taking MATH 201, owns a laptop computer; in symbols,  $P(\text{Taylor})$  is true. We may then use existential generalization to conclude that  $\exists x P(x)$  is true.  $\blacktriangleleft$

**Example 1.5.23**

Write the following argument symbolically and then, using rules of inference, show that the argument is valid.

**SOLUTION** For every real number  $x$ , if  $x$  is an integer, then  $x$  is a rational number. The number  $\sqrt{2}$  is not rational. Therefore,  $\sqrt{2}$  is not an integer.

If we let  $P(x)$  denote the propositional function “ $x$  is an integer” and  $Q(x)$  denote the propositional function “ $x$  is rational,” the argument becomes

$$\begin{array}{c} \forall x \in \mathbf{R} (P(x) \rightarrow Q(x)) \\ \neg Q(\sqrt{2}) \\ \hline \therefore \neg P(\sqrt{2}) \end{array}$$

Since  $\sqrt{2} \in \mathbf{R}$ , we may use universal instantiation to conclude  $P(\sqrt{2}) \rightarrow Q(\sqrt{2})$ . Combining  $P(\sqrt{2}) \rightarrow Q(\sqrt{2})$  and  $\neg Q(\sqrt{2})$ , we may use modus tollens (see Table 1.4.1) to conclude  $\neg P(\sqrt{2})$ . Thus the argument is valid.  $\blacktriangleleft$

The argument in Example 1.5.23 is called **universal modus tollens**.

**Example 1.5.24**

We are given these hypotheses: Everyone loves either Microsoft or Apple. Lynn does not love Microsoft. Show that the conclusion, Lynn loves Apple, follows from the hypotheses.

**SOLUTION** Let  $P(x)$  denote the propositional function “ $x$  loves Microsoft,” and let  $Q(x)$  denote the propositional function “ $x$  loves Apple.” The first hypothesis is  $\forall x(P(x) \vee Q(x))$ . By universal instantiation, we have  $P(\text{Lynn}) \vee Q(\text{Lynn})$ . The second hypothesis is  $\neg P(\text{Lynn})$ . The disjunctive syllogism rule of inference (see Table 1.4.1) now gives  $Q(\text{Lynn})$ , which represents the proposition “Lynn loves Apple.” We conclude that the conclusion does follow from the hypotheses.  $\blacktriangleleft$

## 1.5 Problem-Solving Tips

- To prove that the universally quantified statement  $\forall x P(x)$  is true, show that for *every*  $x$  in the domain of discourse, the proposition  $P(x)$  is true. Showing that  $P(x)$  is true for a *particular* value  $x$  does *not* prove that  $\forall x P(x)$  is true.
- To prove that the existentially quantified statement  $\exists x P(x)$  is true, find *one* value of  $x$  in the domain of discourse for which the proposition  $P(x)$  is true. *One* value suffices.
- To prove that the universally quantified statement  $\forall x P(x)$  is false, find *one* value of  $x$  (a counterexample) in the domain of discourse for which the proposition  $P(x)$  is false.
- To prove that the existentially quantified statement  $\exists x P(x)$  is false, show that for *every*  $x$  in the domain of discourse, the proposition  $P(x)$  is false. Showing that  $P(x)$  is false for a *particular* value  $x$  does *not* prove that  $\exists x P(x)$  is false.

## 1.5 Review Exercises

1. What is a propositional function?
2. What is a domain of discourse?
3. What is a universally quantified statement?
4. What is a counterexample?
5. What is an existentially quantified statement?
6. State the generalized De Morgan's laws for logic.
7. Explain how to prove that a universally quantified statement is true.
8. Explain how to prove that an existentially quantified statement is true.
9. Explain how to prove that a universally quantified statement is false.
10. Explain how to prove that an existentially quantified statement is false.
11. State the universal instantiation rule of inference.
12. State the universal generalization rule of inference.
13. State the existential instantiation rule of inference.
14. State the existential generalization rule of inference.

## 1.5 Exercises

In Exercises 1–6, tell whether the statement is a propositional function. For each statement that is a propositional function, give a domain of discourse.

1.  $(2n + 1)^2$  is an odd integer.
2. Choose an integer between 1 and 10.
3. Let  $x$  be a real number.
4. The movie won the Academy Award as the best picture of 1955.
5.  $1 + 3 = 4$ .
6. There exists  $x$  such that  $x < y$  ( $x, y$  real numbers).

Let  $P(n)$  be the propositional function “ $n$  divides 77.” Write each proposition in Exercises 7–15 in words and tell whether it is true or false. The domain of discourse is  $\mathbb{Z}^+$ .

- |                           |                            |                            |
|---------------------------|----------------------------|----------------------------|
| 7. $P(11)$                | 8. $P(1)$                  | 9. $P(3)$                  |
| 10. $\forall n P(n)$      | 11. $\exists n P(n)$       | 12. $\forall n \neg P(n)$  |
| 13. $\exists n \neg P(n)$ | 14. $\neg(\forall n P(n))$ | 15. $\neg(\exists n P(n))$ |

Let  $P(x)$  be the propositional function “ $x \geq x^2$ .” Tell whether each proposition in Exercises 16–24 is true or false. The domain of discourse is  $\mathbf{R}$ .

- |                            |                           |                            |
|----------------------------|---------------------------|----------------------------|
| 16. $P(1)$                 | 17. $P(2)$                | 18. $P(1/2)$               |
| 19. $\forall x P(x)$       | 20. $\exists x P(x)$      | 21. $\neg(\forall x P(x))$ |
| 22. $\neg(\exists x P(x))$ | 23. $\forall x \neg P(x)$ | 24. $\exists x \neg P(x)$  |

Suppose that the domain of discourse of the propositional function  $P$  is  $\{1, 2, 3, 4\}$ . Rewrite each propositional function in Exercises 25–31 using only negation, disjunction, and conjunction.

- |                                              |                           |                            |
|----------------------------------------------|---------------------------|----------------------------|
| 25. $\forall x P(x)$                         | 26. $\forall x \neg P(x)$ | 27. $\neg(\forall x P(x))$ |
| 28. $\exists x P(x)$                         | 29. $\exists x \neg P(x)$ | 30. $\neg(\exists x P(x))$ |
| 31. $\forall x((x \neq 1) \rightarrow P(x))$ |                           |                            |

Let  $P(x)$  denote the statement “ $x$  is taking a math course.” The domain of discourse is the set of all students. Write each proposition in Exercises 32–37 in words.

- |                                                                                          |                            |
|------------------------------------------------------------------------------------------|----------------------------|
| 32. $\forall x P(x)$                                                                     | 33. $\exists x P(x)$       |
| 34. $\forall x \neg P(x)$                                                                | 35. $\exists x \neg P(x)$  |
| 36. $\neg(\forall x P(x))$                                                               | 37. $\neg(\exists x P(x))$ |
| 38. Write the negation of each proposition in Exercises 32–37 symbolically and in words. |                            |

Let  $P(x)$  denote the statement “ $x$  is a professional athlete,” and let  $Q(x)$  denote the statement “ $x$  plays soccer.” The domain of discourse is the set of all people. Write each proposition in Exercises 39–46 in words. Determine the truth value of each statement.

- |                                                                                          |                                         |
|------------------------------------------------------------------------------------------|-----------------------------------------|
| 39. $\forall x (P(x) \rightarrow Q(x))$                                                  | 40. $\exists x (P(x) \rightarrow Q(x))$ |
| 41. $\forall x (Q(x) \rightarrow P(x))$                                                  | 42. $\exists x (Q(x) \rightarrow P(x))$ |
| 43. $\forall x (P(x) \vee Q(x))$                                                         | 44. $\exists x (P(x) \vee Q(x))$        |
| 45. $\forall x (P(x) \wedge Q(x))$                                                       | 46. $\exists x (P(x) \wedge Q(x))$      |
| 47. Write the negation of each proposition in Exercises 39–46 symbolically and in words. |                                         |

Let  $P(x)$  denote the statement “ $x$  is an accountant,” and let  $Q(x)$  denote the statement “ $x$  owns a Porsche.” Write each statement in Exercises 48–51 symbolically.

- |                                                                                          |  |
|------------------------------------------------------------------------------------------|--|
| 48. All accountants own Porsches.                                                        |  |
| 49. Some accountant owns a Porsche.                                                      |  |
| 50. All owners of Porsches are accountants.                                              |  |
| 51. Someone who owns a Porsche is an accountant.                                         |  |
| 52. Write the negation of each proposition in Exercises 48–51 symbolically and in words. |  |

Determine the truth value of each statement in Exercises 53–58. The domain of discourse is  $\mathbf{R}$ . Justify your answers.

- |                          |                          |
|--------------------------|--------------------------|
| 53. $\forall x(x^2 > x)$ | 54. $\exists x(x^2 > x)$ |
|--------------------------|--------------------------|

55.  $\forall x(x > 1 \rightarrow x^2 > x)$   
 56.  $\exists x(x > 1 \rightarrow x^2 > x)$   
 57.  $\forall x(x > 1 \rightarrow x/(x^2 + 1) < 1/3)$   
 58.  $\exists x(x > 1 \rightarrow x/(x^2 + 1) < 1/3)$   
 59. Write the negation of each proposition in Exercises 53–58 symbolically and in words.  
 60. Could the pseudocode of Example 1.5.7 be written as follows?

```
for i = 1 to n
 if ($\neg P(d_i)$)
 return false
 else
 return true
```

What is the literal meaning of each statement in Exercises 61–71? What is the intended meaning? Clarify each statement by rephrasing it and writing it symbolically.

61. From *Dear Abby*: All men do not cheat on their wives.  
 62. From the *San Antonio Express-News*: All old things don't covet twenty-somethings.  
 63. All 74 hospitals did not report every month.  
 64. Economist Robert J. Samuelson: Every environmental problem is not a tragedy.  
 65. Comment from a Door County alderman: This is still Door County and we all don't have a degree.  
 66. Headline over a Martha Stewart column: All lampshades can't be cleaned.  
 67. Headline in the *New York Times*: A World Where All Is Not Sweetness and Light.  
 68. Headline over a story about subsidized housing: Everyone can't afford home.  
 69. George W. Bush: I understand everybody in this country doesn't agree with the decisions I've made.  
 70. From *Newsweek*: Formal investigations are a sound practice in the right circumstances, but every circumstance is not right.  
 71. Joe Girardi (manager of the New York Yankees): Every move is not going to work out.

72. (a) Use a truth table to prove that if  $p$  and  $q$  are propositions, at least one of  $p \rightarrow q$  or  $q \rightarrow p$  is true.  
 (b) Let  $I(x)$  be the propositional function “ $x$  is an integer” and let  $P(x)$  be the propositional function “ $x$  is a positive number.” The domain of discourse is  $\mathbf{R}$ . Determine whether or not the following proof that all integers are positive or all positive real numbers are integers is correct.

By part (a),

$$\forall x ((I(x) \rightarrow P(x)) \vee (P(x) \rightarrow I(x)))$$

is true. In words: For all  $x$ , if  $x$  is an integer, then  $x$  is positive; or if  $x$  is positive, then  $x$  is an integer. Therefore, all integers are positive or all positive real numbers are integers.

73. Prove Theorem 1.5.14, part (b).  
 74. Analyze the following comments by film critic Roger Ebert: No good movie is too long. No bad movie is short enough. *Love Actually* is good, but it is too long.  
 75. Which rule of inference is used in the following argument? Every rational number is of the form  $p/q$ , where  $p$  and  $q$  are integers. Therefore, 9.345 is of the form  $p/q$ .

In Exercises 76–78, give an argument using rules of inference to show that the conclusion follows from the hypotheses.

76. Hypotheses: Everyone in the class has a graphing calculator. Everyone who has a graphing calculator understands the trigonometric functions. Conclusion: Ralphie, who is in the class, understands the trigonometric functions.  
 77. Hypotheses: Ken, a member of the Titans, can hit the ball a long way. Everyone who can hit the ball a long way can make a lot of money. Conclusion: Some member of the Titans can make a lot of money.  
 78. Hypotheses: Everyone in the discrete mathematics class loves proofs. Someone in the discrete mathematics class has never taken calculus. Conclusion: Someone who loves proofs has never taken calculus.  
 79. Show that universal generalization (see Table 1.5.1) is valid.  
 80. Show that existential instantiation (see Table 1.5.1) is valid.  
 81. Show that existential generalization (see Table 1.5.1) is valid.

## 1.6 Nested Quantifiers

Consider writing the statement

The sum of any two positive real numbers is positive,

symbolically. We first note that since two numbers are involved, we will need two variables, say  $x$  and  $y$ . The assertion can be restated as: If  $x > 0$  and  $y > 0$ , then  $x + y > 0$ . The given statement says that the sum of *any* two positive real numbers is positive, so we need two universal quantifiers. If we let  $P(x, y)$  denote the expression  $(x > 0) \wedge (y > 0) \rightarrow (x + y > 0)$ , the given statement can be written symbolically as

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

In words, for every  $x$  and for every  $y$ , if  $x > 0$  and  $y > 0$ , then  $x + y > 0$ . The domain of discourse of the two-variable propositional function  $P$  is  $\mathbf{R} \times \mathbf{R}$ , which means that each variable  $x$  and  $y$  must belong to the set of real numbers. Multiple quantifiers such as  $\forall x \forall y$  are said to be **nested quantifiers**. In this section we explore nested quantifiers in detail.

### Example 1.6.1

Restate  $\forall m \exists n (m < n)$  in words. The domain of discourse is the set  $\mathbf{Z} \times \mathbf{Z}$ .

**SOLUTION** We may first rephrase this statement as: For every  $m$ , there exists  $n$  such that  $m < n$ . Less formally, this means that if you take any integer  $m$  whatsoever, there is an integer  $n$  greater than  $m$ . Another restatement is then: There is no greatest integer. ▲

### Example 1.6.2

Write the assertion

Everybody loves somebody,

symbolically, letting  $L(x, y)$  be the statement “ $x$  loves  $y$ .”

**SOLUTION** “Everybody” requires universal quantification and “somebody” requires existential quantification. Thus, the given statement may be written symbolically as

$$\forall x \exists y L(x, y).$$

In words, for every person  $x$ , there exists a person  $y$  such that  $x$  loves  $y$ .

Notice that

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

is *not* a correct interpretation of the original statement. This latter statement is: There exists a person  $x$  such that for all  $y$ ,  $x$  loves  $y$ . Less formally, someone loves everyone. The order of quantifiers is important; changing the order can change the meaning. ▲

By definition, the statement  $\forall x \forall y P(x, y)$ , with domain of discourse  $X \times Y$ , is true if, for *every*  $x \in X$  and for *every*  $y \in Y$ ,  $P(x, y)$  is true. The statement  $\forall x \forall y P(x, y)$  is false if there is *at least one*  $x \in X$  and *at least one*  $y \in Y$  such that  $P(x, y)$  is false.

### Example 1.6.3

Consider the statement

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

The domain of discourse is  $\mathbf{R} \times \mathbf{R}$ . This statement is true because, for every real number  $x$  and for every real number  $y$ , the conditional proposition

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

is true. In words, for every real number  $x$  and for every real number  $y$ , if  $x$  and  $y$  are positive, their sum is positive. ▲

### Example 1.6.4

Consider the statement

$$\forall x \forall y ((x > 0) \wedge (y < 0) \rightarrow (x + y \neq 0)).$$

The domain of discourse is  $\mathbf{R} \times \mathbf{R}$ . This statement is false because if  $x = 1$  and  $y = -1$ , the conditional proposition

$$(x > 0) \wedge (y < 0) \rightarrow (x + y \neq 0)$$

is false. We say that the pair  $x = 1$  and  $y = -1$  is a counterexample. ▲

**Example 1.6.5**

Suppose that  $P$  is a propositional function with domain of discourse  $\{d_1, \dots, d_n\} \times \{d_1, \dots, d_n\}$ . The following pseudocode determines whether  $\forall x \forall y P(x, y)$  is true or false:

```

for i = 1 to n
 for j = 1 to n
 if ($\neg P(d_i, d_j)$)
 return false
 return true

```

The for loops examine members of the domain of discourse. If they find a pair  $d_i, d_j$  for which  $P(d_i, d_j)$  is false, the condition  $\neg P(d_i, d_j)$  in the if statement is true; so the code returns false [to indicate that  $\forall x \forall y P(x, y)$  is false] and terminates. In this case, the pair  $d_i, d_j$  is a counterexample. If  $P(d_i, d_j)$  is true for every pair  $d_i, d_j$ , the condition  $\neg P(d_i, d_j)$  in the if statement is always false. In this case, the for loops run to completion, after which the code returns true [to indicate that  $\forall x \forall y P(x, y)$  is true] and terminates.  $\blacktriangleleft$

By definition, the statement  $\forall x \exists y P(x, y)$ , with domain of discourse  $X \times Y$ , is true if, for every  $x \in X$ , there is *at least one*  $y \in Y$  for which  $P(x, y)$  is true. The statement  $\forall x \exists y P(x, y)$  is false if there is *at least one*  $x \in X$  such that  $P(x, y)$  is false for *every*  $y \in Y$ .

**Example 1.6.6**

Consider the statement

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

The domain of discourse is  $\mathbf{R} \times \mathbf{R}$ . This statement is true because, for every real number  $x$ , there is at least one  $y$  (namely  $y = -x$ ) for which  $x + y = 0$  is true. In words, for every real number  $x$ , there is a number that when added to  $x$  makes the sum zero.  $\blacktriangleleft$

**Example 1.6.7**

Consider the statement

$$\forall x \exists y (x > y).$$

The domain of discourse is  $\mathbf{Z}^+ \times \mathbf{Z}^+$ . This statement is false because there is at least one  $x$ , namely  $x = 1$ , such that  $x > y$  is false for every positive integer  $y$ .  $\blacktriangleleft$

**Example 1.6.8**

Suppose that  $P$  is a propositional function with domain of discourse  $\{d_1, \dots, d_n\} \times \{d_1, \dots, d_n\}$ . The following pseudocode determines whether  $\forall x \exists y P(x, y)$  is true or false:

```

for i = 1 to n
 if ($\neg \text{exists_dj}(i)$)
 return false
 return true
 exists_dj(i) {
 for j = 1 to n
 if ($P(d_i, d_j)$)
 return true
 return false
 }

```

If for each  $d_i$ , there exists  $d_j$  such that  $P(d_i, d_j)$  is true, then for each  $i$ ,  $P(d_i, d_j)$  is true for some  $j$ . Thus,  $\text{exists\_dj}(i)$  returns true for every  $i$ . Since  $\neg \text{exists\_dj}(i)$  is always false, the first for loop eventually terminates and true is returned to indicate that  $\forall x \exists y P(x, y)$  is true.

If for some  $d_i$ ,  $P(d_i, d_j)$  is false for every  $j$ , then, for this  $i$ ,  $P(d_i, d_j)$  is false for every  $j$ . In this case, the for loop in `exists_dj(i)` runs to termination and false is returned. Since  $\neg \text{exists\_dj}(i)$  is true, false is returned to indicate that  $\forall x \exists y P(x, y)$  is false. ▲

By definition, the statement  $\exists x \forall y P(x, y)$ , with domain of discourse  $X \times Y$ , is true if there is *at least one*  $x \in X$  such that  $P(x, y)$  is true for *every*  $y \in Y$ . The statement  $\exists x \forall y P(x, y)$  is false if, for *every*  $x \in X$ , there is *at least one*  $y \in Y$  such that  $P(x, y)$  is false.

### Example 1.6.9

Consider the statement  $\exists x \forall y (x \leq y)$ . The domain of discourse is  $\mathbf{Z}^+ \times \mathbf{Z}^+$ . This statement is true because there is at least one positive integer  $x$  (namely  $x = 1$ ) for which  $x \leq y$  is true for every positive integer  $y$ . In words, there is a smallest positive integer (namely 1). ▲

### Example 1.6.10

Consider the statement  $\exists x \forall y (x \geq y)$ . The domain of discourse is  $\mathbf{Z}^+ \times \mathbf{Z}^+$ . This statement is false because, for every positive integer  $x$ , there is at least one positive integer  $y$ , namely  $y = x + 1$ , such that  $x \geq y$  is false. In words, there is no greatest positive integer. ▲

By definition, the statement  $\exists x \exists y P(x, y)$ , with domain of discourse  $X \times Y$ , is true if there is *at least one*  $x \in X$  and *at least one*  $y \in Y$  such that  $P(x, y)$  is true. The statement  $\exists x \exists y P(x, y)$  is false if, for *every*  $x \in X$  and for *every*  $y \in Y$ ,  $P(x, y)$  is false.

### Example 1.6.11

Consider the statement

$$\exists x \exists y ((x > 1) \wedge (y > 1) \wedge (xy = 6)).$$

The domain of discourse is  $\mathbf{Z}^+ \times \mathbf{Z}^+$ . This statement is true because there is at least one integer  $x > 1$  (namely  $x = 2$ ) and at least one integer  $y > 1$  (namely  $y = 3$ ) such that  $xy = 6$ . In words, 6 is composite (i.e., not prime). ▲

### Example 1.6.12

Consider the statement

$$\exists x \exists y ((x > 1) \wedge (y > 1) \wedge (xy = 7)).$$

The domain of discourse is  $\mathbf{Z}^+ \times \mathbf{Z}^+$ . This statement is false because for every positive integer  $x$  and for every positive integer  $y$ ,

$$(x > 1) \wedge (y > 1) \wedge (xy = 7)$$

is false. In words, 7 is prime. ▲

The generalized De Morgan's laws for logic (Theorem 1.5.14) can be used to negate a proposition containing nested quantifiers.

### Example 1.6.13

Using the generalized De Morgan's laws for logic, we find that the negation of  $\forall x \exists y P(x, y)$  is

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

Notice how in the negation,  $\forall$  and  $\exists$  are interchanged. ▲

### Example 1.6.14

Write the negation of  $\exists x \forall y (xy < 1)$ , where the domain of discourse is  $\mathbf{R} \times \mathbf{R}$ . Determine the truth value of the given statement and its negation.

**SOLUTION** Using the generalized De Morgan's laws for logic, we find that the negation is

$$\neg(\exists x \forall y(xy < 1)) \equiv \forall x \neg(\forall y(xy < 1)) \equiv \forall x \exists y \neg(xy < 1) \equiv \forall x \exists y(xy \geq 1).$$

The given statement  $\exists x \forall y(xy < 1)$  is true because there is at least one  $x$  (namely  $x = 0$ ) such that  $xy < 1$  for every  $y$ . Since the given statement is true, its negation is false.  $\blacktriangleleft$

We conclude with a logic game, which presents an alternative way to determine whether a quantified propositional function is true or false. André Berthiaume contributed this example.

### Example 1.6.15

**The Logic Game** Given a quantified propositional function such as  $\forall x \exists y P(x, y)$ , you and your opponent, whom we call Farley, play a logic game. Your goal is to try to make  $P(x, y)$  true, and Farley's goal is to try to make  $P(x, y)$  false. The game begins with the first (left) quantifier. If the quantifier is  $\forall$ , Farley chooses a value for that variable; if the quantifier is  $\exists$ , you choose a value for that variable. The game continues with the second quantifier. After values are chosen for all the variables, if  $P(x, y)$  is true, you win; if  $P(x, y)$  is false, Farley wins. We will show that if you can always win regardless of how Farley chooses values for the variables, the quantified statement is true, but if Farley can choose values for the variables so that you cannot win, the quantified statement is false.

Consider the statement

$$\forall x \exists y(x + y = 0). \quad (1.6.1)$$

The domain of discourse is  $\mathbf{R} \times \mathbf{R}$ . Since the first quantifier is  $\forall$ , Farley goes first and chooses a value for  $x$ . Since the second quantifier is  $\exists$ , you go second. Regardless of what value Farley chose, you can choose  $y = -x$ , which makes the statement  $x + y = 0$  true. You can always win the game, so the statement (1.6.1) is true.

Next, consider the statement

$$\exists x \forall y(x + y = 0). \quad (1.6.2)$$

Again, the domain of discourse is  $\mathbf{R} \times \mathbf{R}$ . Since the first quantifier is  $\exists$ , you go first and choose a value for  $x$ . Since the second quantifier is  $\forall$ , Farley goes second. Regardless of what value you chose, Farley can always choose a value for  $y$ , which makes the statement  $x + y = 0$  false. (If you choose  $x = 0$ , Farley can choose  $y = 1$ . If you choose  $x \neq 0$ , Farley can choose  $y = 0$ .) Farley can always win the game, so the statement (1.6.2) is false.

We discuss why the game correctly determines the truth value of a quantified propositional function. Consider  $\forall x \forall y P(x, y)$ . If Farley can always win the game, this means that Farley can find values for  $x$  and  $y$  that make  $P(x, y)$  false. In this case, the propositional function is false; the values Farley found provide a counterexample. If Farley cannot win the game, no counterexample exists; in this case, the propositional function is true.

Consider  $\forall x \exists y P(x, y)$ . Farley goes first and chooses a value for  $x$ . You choose second. If, no matter what value Farley chose, you can choose a value for  $y$  that makes  $P(x, y)$  true, you can always win the game and the propositional function is true. However, if Farley can choose a value for  $x$  so that every value you choose for  $y$  makes  $P(x, y)$  false, then you will always lose the game and the propositional function is false.

An analysis of the other cases also shows that if you can always win the game, the propositional function is true; but if Farley can always win the game, the propositional function is false.

The logic game extends to propositional functions of more than two variables. The rules are the same and, again, if you can always win the game, the propositional function is true; but if Farley can always win the game, the propositional function is false.  $\blacktriangleleft$

## 1.6 Problem-Solving Tips

- To prove that  $\forall x \forall y P(x, y)$  is true, where the domain of discourse is  $X \times Y$ , you must show that  $P(x, y)$  is true for all values of  $x \in X$  and  $y \in Y$ . One technique is to argue that  $P(x, y)$  is true using the symbols  $x$  and  $y$  to stand for *arbitrary* elements in  $X$  and  $Y$ .
- To prove that  $\forall x \forall y P(x, y)$  is false, where the domain of discourse is  $X \times Y$ , find one value of  $x \in X$  and one value of  $y \in Y$  (*two* values suffice—one for  $x$  and one for  $y$ ) that make  $P(x, y)$  false.
- To prove that  $\forall x \exists y P(x, y)$  is true, where the domain of discourse is  $X \times Y$ , you must show that for all  $x \in X$ , there is at least one  $y \in Y$  such that  $P(x, y)$  is true. One technique is to let  $x$  stand for an arbitrary element in  $X$  and then find a value for  $y \in Y$  (*one* value suffices!) that makes  $P(x, y)$  true.
- To prove that  $\forall x \exists y P(x, y)$  is false, where the domain of discourse is  $X \times Y$ , you must show that for at least one  $x \in X$ ,  $P(x, y)$  is false for every  $y \in Y$ . One technique is to find a value of  $x \in X$  (again *one* value suffices!) that has the property that  $P(x, y)$  is false for every  $y \in Y$ . Having chosen a value for  $x$ , let  $y$  stand for an arbitrary element of  $Y$  and show that  $P(x, y)$  is always false.
- To prove that  $\exists x \forall y P(x, y)$  is true, where the domain of discourse is  $X \times Y$ , you must show that for at least one  $x \in X$ ,  $P(x, y)$  is true for every  $y \in Y$ . One technique is to find a value of  $x \in X$  (again *one* value suffices!) that has the property that  $P(x, y)$  is true for every  $y \in Y$ . Having chosen a value for  $x$ , let  $y$  stand for an arbitrary element of  $Y$  and show that  $P(x, y)$  is always true.
- To prove that  $\exists x \forall y P(x, y)$  is false, where the domain of discourse is  $X \times Y$ , you must show that for all  $x \in X$ , there is at least one  $y \in Y$  such that  $P(x, y)$  is false. One technique is to let  $x$  stand for an arbitrary element in  $X$  and then find a value for  $y \in Y$  (*one* value suffices!) that makes  $P(x, y)$  false.
- To prove that  $\exists x \exists y P(x, y)$  is true, where the domain of discourse is  $X \times Y$ , find one value of  $x \in X$  and one value of  $y \in Y$  (*two* values suffice—one for  $x$  and one for  $y$ ) that make  $P(x, y)$  true.
- To prove that  $\exists x \exists y P(x, y)$  is false, where the domain of discourse is  $X \times Y$ , you must show that  $P(x, y)$  is false for all values of  $x \in X$  and  $y \in Y$ . One technique is to argue that  $P(x, y)$  is false using the symbols  $x$  and  $y$  to stand for *arbitrary* elements in  $X$  and  $Y$ .
- To negate an expression with nested quantifiers, use the generalized De Morgan's laws for logic. Loosely speaking,  $\forall$  and  $\exists$  are interchanged. Don't forget that the negation of  $p \rightarrow q$  is equivalent to  $p \wedge \neg q$ .

## 1.6 Review Exercises

1. What is the interpretation of  $\forall x \forall y P(x, y)$ ? When is this quantified expression true? When is it false?
2. What is the interpretation of  $\forall x \exists y P(x, y)$ ? When is this quantified expression true? When is it false?
3. What is the interpretation of  $\exists x \forall y P(x, y)$ ? When is this quantified expression true? When is it false?
4. What is the interpretation of  $\exists x \exists y P(x, y)$ ? When is this quantified expression true? When is it false?
5. Give an example to show that, in general,  $\forall x \exists y P(x, y)$  and  $\exists x \forall y P(x, y)$  have different meanings.
6. Write the negation of  $\forall x \forall y P(x, y)$  using the generalized De Morgan's laws for logic.

7. Write the negation of  $\forall x \exists y P(x, y)$  using the generalized De Morgan's laws for logic.
8. Write the negation of  $\exists x \forall y P(x, y)$  using the generalized De Morgan's laws for logic.
9. Write the negation of  $\exists x \exists y P(x, y)$  using the generalized De Morgan's laws for logic.
10. Explain the rules for playing the logic game. How can the logic game be used to determine the truth value of a quantified expression?

## 1.6 Exercises

In Exercises 1–33, the set  $D_1$  consists of three students: Garth, who is 5 feet 11 inches tall; Erin, who is 5 feet 6 inches tall; and Marty, who is 6 feet tall. The set  $D_2$  consists of four students: Dale, who is 6 feet tall; Garth, who is 5 feet 11 inches tall; Erin, who is 5 feet 6 inches tall; and Marty, who is 6 feet tall. The set  $D_3$  consists of one student: Dale, who is 6 feet tall. The set  $D_4$  consists of three students: Pat, Sandy, and Gale, each of whom is 5 feet 11 inches tall.

In Exercises 1–21,  $T_1(x, y)$  is the propositional function “ $x$  is taller than  $y$ .” Write each proposition in Exercises 1–4 in words.

1.  $\forall x \forall y T_1(x, y)$
2.  $\forall x \exists y T_1(x, y)$
3.  $\exists x \forall y T_1(x, y)$
4.  $\exists x \exists y T_1(x, y)$
5. Write the negation of each proposition in Exercises 1–4 in words and symbolically.

In Exercises 6–21, tell whether each proposition in Exercises 1–4 is true or false if the domain of discourse is  $D_i \times D_j$  for the given values of  $i$  and  $j$ .

6.  $i = 1, j = 1$
7.  $i = 1, j = 2$
8.  $i = 1, j = 3$
9.  $i = 1, j = 4$
10.  $i = 2, j = 1$
11.  $i = 2, j = 2$
12.  $i = 2, j = 3$
13.  $i = 2, j = 4$
14.  $i = 3, j = 1$
15.  $i = 3, j = 2$
16.  $i = 3, j = 3$
17.  $i = 3, j = 4$
18.  $i = 4, j = 1$
19.  $i = 4, j = 2$
20.  $i = 4, j = 3$
21.  $i = 4, j = 4$

In Exercises 22–27,  $T_2(x, y)$  is the propositional function “ $x$  is taller than or the same height as  $y$ .” Write each proposition in Exercises 22–25 in words.

22.  $\forall x \forall y T_2(x, y)$
23.  $\forall x \exists y T_2(x, y)$
24.  $\exists x \forall y T_2(x, y)$
25.  $\exists x \exists y T_2(x, y)$
26. Write the negation of each proposition in Exercises 22–25 in words and symbolically.
27. Tell whether each proposition in Exercises 22–25 is true or false if the domain of discourse is  $D_i \times D_j$  for each pair of values  $i, j$  given in Exercises 6–21. The sets  $D_1, \dots, D_4$  are defined before Exercise 1.

In Exercises 28–33,  $T_3(x, y)$  is the propositional function “if  $x$  and  $y$  are distinct persons, then  $x$  is taller than  $y$ .” Write each proposition in Exercises 28–31 in words.

28.  $\forall x \forall y T_3(x, y)$
29.  $\forall x \exists y T_3(x, y)$
30.  $\exists x \forall y T_3(x, y)$
31.  $\exists x \exists y T_3(x, y)$

32. Write the negation of each proposition in Exercises 28–31 in words and symbolically.
33. Tell whether each proposition in Exercises 28–31 is true or false if the domain of discourse is  $D_i \times D_j$  for each pair of values  $i, j$  given in Exercises 6–21. The sets  $D_1, \dots, D_4$  are defined before Exercise 1.

Let  $L(x, y)$  be the propositional function “ $x$  loves  $y$ .” The domain of discourse is the Cartesian product of the set of all living people with itself (i.e., both  $x$  and  $y$  take on values in the set of all living people). Write each proposition in Exercises 34–37 symbolically. Which do you think are true?

34. Someone loves everybody.
35. Everybody loves everybody.
36. Somebody loves somebody.
37. Everybody loves somebody.
38. Write the negation of each proposition in Exercises 34–37 in words and symbolically.

Let  $A(x, y)$  be the propositional function “ $x$  attended  $y$ ’s office hours” and let  $E(x)$  be the propositional function “ $x$  is enrolled in a discrete math class.” Let  $S$  be the set of students and let  $T$  denote the set of teachers—all at Hudson University. The domain of discourse of  $A$  is  $S \times T$  and the domain of discourse of  $E$  is  $S$ . Write each proposition in Exercises 39–42 symbolically.

39. Brit attended someone’s office hours.
40. No one attended Professor Sandwich’s office hours.
41. Every discrete math student attended someone’s office hours.
42. All teachers had at least one student attend their office hours.

Let  $P(x, y)$  be the propositional function  $x \geq y$ . The domain of discourse is  $\mathbf{Z}^+ \times \mathbf{Z}^+$ . Tell whether each proposition in Exercises 43–46 is true or false.

43.  $\forall x \forall y P(x, y)$
44.  $\forall x \exists y P(x, y)$
45.  $\exists x \forall y P(x, y)$
46.  $\exists x \exists y P(x, y)$
47. Write the negation of each proposition in Exercises 43–46.

Determine the truth value of each statement in Exercises 48–65. The domain of discourse is  $\mathbf{R} \times \mathbf{R}$ . Justify your answers.

48.  $\forall x \forall y (x^2 < y + 1)$
49.  $\forall x \exists y (x^2 < y + 1)$
50.  $\exists x \forall y (x^2 < y + 1)$
51.  $\exists x \exists y (x^2 < y + 1)$
52.  $\exists y \forall x (x^2 < y + 1)$
53.  $\forall y \exists x (x^2 < y + 1)$
54.  $\forall x \forall y (x^2 + y^2 = 9)$
55.  $\forall x \exists y (x^2 + y^2 = 9)$

## 56 Chapter 1 ◆ Sets and Logic

56.  $\exists x \forall y (x^2 + y^2 = 9)$       57.  $\exists x \exists y (x^2 + y^2 = 9)$

58.  $\forall x \forall y (x^2 + y^2 \geq 0)$

59.  $\forall x \exists y (x^2 + y^2 \geq 0)$

60.  $\exists x \forall y (x^2 + y^2 \geq 0)$

61.  $\exists x \exists y (x^2 + y^2 \geq 0)$

62.  $\forall x \forall y ((x < y) \rightarrow (x^2 < y^2))$

63.  $\forall x \exists y ((x < y) \rightarrow (x^2 < y^2))$

64.  $\exists x \forall y ((x < y) \rightarrow (x^2 < y^2))$

65.  $\exists x \exists y ((x < y) \rightarrow (x^2 < y^2))$

66. Write the negation of each proposition in Exercises 48–65.

67. Suppose that  $P$  is a propositional function with domain of discourse  $\{d_1, \dots, d_n\} \times \{d_1, \dots, d_n\}$ . Write pseudocode that determines whether

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

is true or false.

68. Suppose that  $P$  is a propositional function with domain of discourse  $\{d_1, \dots, d_n\} \times \{d_1, \dots, d_n\}$ . Write pseudocode that determines whether

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

is true or false.

69. Explain how the logic game (Example 1.6.15) determines whether each proposition in Exercises 48–65 is true or false.

70. Use the logic game (Example 1.6.15) to determine whether the proposition

$$\forall x \forall y \exists z ((z > x) \wedge (z < y))$$

is true or false. The domain of discourse is  $\mathbf{Z} \times \mathbf{Z} \times \mathbf{Z}$ .

71. Use the logic game (Example 1.6.15) to determine whether the proposition

$$\forall x \forall y \exists z ((z < x) \wedge (z < y))$$

is true or false. The domain of discourse is  $\mathbf{Z} \times \mathbf{Z} \times \mathbf{Z}$ .

72. Use the logic game (Example 1.6.15) to determine whether the proposition

$$\forall x \forall y \exists z ((x < y) \rightarrow ((z > x) \wedge (z < y)))$$

is true or false. The domain of discourse is  $\mathbf{Z} \times \mathbf{Z} \times \mathbf{Z}$ .

73. Use the logic game (Example 1.6.15) to determine whether the proposition

$$\forall x \forall y \exists z ((x < y) \rightarrow ((z > x) \wedge (z < y)))$$

is true or false. The domain of discourse is  $\mathbf{R} \times \mathbf{R} \times \mathbf{R}$ .

Assume that  $\forall x \forall y P(x, y)$  is true and that the domain of discourse is nonempty. Which of Exercises 74–76 must also be true? Prove your answer.

74.  $\forall x \exists y P(x, y)$       75.  $\exists x \forall y P(x, y)$       76.  $\exists x \exists y P(x, y)$

Assume that  $\exists x \forall y P(x, y)$  is true and that the domain of discourse is nonempty. Which of Exercises 77–79 must also be true? Prove your answer.

77.  $\forall x \forall y P(x, y)$       78.  $\forall x \exists y P(x, y)$       79.  $\exists x \exists y P(x, y)$

Assume that  $\exists x \exists y P(x, y)$  is true and that the domain of discourse is nonempty. Which of Exercises 80–82 must also be true? Prove your answer.

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

81.  $\forall x \exists y P(x, y)$

82.  $\exists x \forall y P(x, y)$

Assume that  $\forall x \forall y P(x, y)$  is false and that the domain of discourse is nonempty. Which of Exercises 83–85 must also be false? Prove your answer.

83.  $\forall x \exists y P(x, y)$       84.  $\exists x \forall y P(x, y)$       85.  $\exists x \exists y P(x, y)$

Assume that  $\forall x \exists y P(x, y)$  is false and that the domain of discourse is nonempty. Which of Exercises 86–88 must also be false? Prove your answer.

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

87.  $\exists x \forall y P(x, y)$

88.  $\exists x \exists y P(x, y)$

Assume that  $\exists x \forall y P(x, y)$  is false and that the domain of discourse is nonempty. Which of Exercises 89–91 must also be false? Prove your answer.

89.  $\forall x \forall y P(x, y)$       90.  $\forall x \exists y P(x, y)$       91.  $\exists x \exists y P(x, y)$

Assume that  $\exists x \exists y P(x, y)$  is false and that the domain of discourse is nonempty. Which of Exercises 92–94 must also be false? Prove your answer.

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

93.  $\forall x \exists y P(x, y)$

94.  $\exists x \forall y P(x, y)$

Which of Exercises 95–98 is logically equivalent to  $\neg(\forall x \exists y P(x, y))$ ? Explain.

95.  $\exists x \neg(\forall y P(x, y))$

96.  $\forall x \neg(\exists y P(x, y))$

97.  $\exists x \forall y \neg P(x, y)$

98.  $\exists x \exists y \neg P(x, y)$

99. [Requires calculus] The definition of

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

is: For every  $\varepsilon > 0$ , there exists  $\delta > 0$  such that for all  $x$  if  $0 < |x - a| < \delta$ , then  $|f(x) - L| < \varepsilon$ . Write this definition symbolically using  $\forall$  and  $\exists$ .

100. [Requires calculus] Write the negation of the definition of limit (see Exercise 99) in words and symbolically using  $\forall$  and  $\exists$  but not  $\neg$ .

\*101. [Requires calculus] Write the definition of “ $\lim_{x \rightarrow a} f(x)$  does not exist” (see Exercise 99) in words and symbolically using  $\forall$  and  $\exists$  but not  $\neg$ .

102. Consider the headline: Every school may not be right for every child. What is the literal meaning? What is the intended meaning? Clarify the headline by rephrasing it and writing it symbolically.

## Problem-Solving Corner

### Problem

Assume that  $\forall x \exists y P(x, y)$  is true and that the domain of discourse is nonempty. Which of the following must also be true? If the statement is true, explain; otherwise, give a counterexample.

- (a)  $\forall x \forall y P(x, y)$
- (b)  $\exists x \forall y P(x, y)$
- (c)  $\exists x \exists y P(x, y)$

### Attacking the Problem

Let's begin with part (a). We are given that  $\forall x \exists y P(x, y)$  is true, which says, in words, for every  $x$ , there exists at least one  $y$  for which  $P(x, y)$  is true. If (a) is also true, then, in words, for every  $x$ , *for every*  $y$ ,  $P(x, y)$  is true. Let the words sink in. If for every  $x$ ,  $P(x, y)$  is true for *at least one*  $y$ , doesn't it seem unlikely that it would follow that  $P(x, y)$  is true for *every*  $y$ ? We suspect that (a) could be false. We'll need to come up with a counterexample.

Contrasting statement (b) with the given statement, we see that the quantifiers  $\forall$  and  $\exists$  have been swapped. There is a difference. In the given true statement  $\forall x \exists y P(x, y)$ , given *any*  $x$ , it's possible to find a  $y$ , which may depend on  $x$ , that makes  $P(x, y)$  true. For statement (b),  $\exists x \forall y P(x, y)$ , to be true, for some  $x$ ,  $P(x, y)$  would need to be true for every  $y$ . Again, let the words sink in. These two statements seem quite different. We suspect that (b) also could be false. Again, we'll need to come up with a counterexample.

Now let's turn to part (c). We are given that  $\forall x \exists y P(x, y)$  is true, which says, in words, for every  $x$ , there exists at least one  $y$  for which  $P(x, y)$  is true. For statement (c),  $\exists x \exists y P(x, y)$ , to be true, for some  $x$  and for some  $y$ ,  $P(x, y)$  must be true. But the given statement says that for *every*  $x$ , there exists at least one  $y$  for which  $P(x, y)$  is true. So if we pick one  $x$  (and we know we can since the domain of discourse is nonempty), the given statement assures us that there exists at least one  $y$  for which  $P(x, y)$  is true. Thus part (c) must be true. In fact, we have just given an explanation!

## Quantifiers

### Finding a Solution

As noted, we have already solved part (c). We need counterexamples for parts (a) and (b).

For part (a), we need the given statement,  $\forall x \exists y P(x, y)$ , to be true and  $\forall x \forall y P(x, y)$  to be false. In order for the given statement to be true, we must find a propositional function  $P(x, y)$  satisfying

$$\text{for every } x, \text{ there exists } y \text{ such that } P(x, y) \text{ is true.} \quad (1)$$

In order for (a) to be false, we must have

$$\text{at least one value of } x \text{ and at least one value of } y \\ \text{such that } P(x, y) \text{ is false.} \quad (2)$$

We can arrange for (1) and (2) to hold simultaneously if we choose  $P(x, y)$  so that for every  $x$ ,  $P(x, y)$  is true for some  $y$ , *but* for at least one  $x$ ,  $P(x, y)$  is also false for some other value of  $y$ . Upon reflection, many mathematical statements have this property. For example,  $x > y$ ,  $x, y \in \mathbf{R}$ , suffices. For every  $x$ , there exists  $y$  such that  $x > y$  is true. Furthermore, for *every*  $x$  (and, in particular, for at least one value of  $x$ ), there exists  $y$  such that  $x > y$  is false.

For part (b), we again need the given statement,  $\forall x \exists y P(x, y)$ , to be true and  $\exists x \forall y P(x, y)$  to be false. In order for the given statement to be true, we must find a propositional function  $P(x, y)$  satisfying (1). In order for (b) to be false, we must have

$$\text{for every } x, \text{ there exists at least one value of } y \text{ such} \\ \text{that } P(x, y) \text{ is false.} \quad (3)$$

We can arrange for (1) and (3) to hold simultaneously if we choose  $P(x, y)$  so that for every  $x$ ,  $P(x, y)$  is true for some  $y$  *and* false for some other value of  $y$ . We noted in the preceding paragraph that  $x > y$ ,  $x, y \in \mathbf{R}$ , has this property.

### Formal Solution

- (a) We give an example to show that statement (a) can be false while the given statement is true. Let  $P(x, y)$  be the propositional function  $x > y$  with domain of discourse  $\mathbf{R} \times \mathbf{R}$ . Then  $\forall x \exists y P(x, y)$  is true since for any  $x$ , we may choose  $y = x - 1$  to make  $P(x, y)$  true. At the same time,  $\forall x \forall y P(x, y)$  is false. A counterexample is  $x = 0, y = 1$ .

- (b) We give an example to show that statement (b) can be false while the given statement is true. Let  $P(x, y)$  be the propositional function  $x > y$  with domain of discourse  $\mathbf{R} \times \mathbf{R}$ . As we showed in part (a),  $\forall x \exists y P(x, y)$  is true. Now we show that  $\exists x \forall y P(x, y)$  is false. Let  $x$  be an arbitrary element in  $\mathbf{R}$ . We may choose  $y = x + 1$  to make  $x > y$  false. Thus for every  $x$ , there exists  $y$  such that  $P(x, y)$  is false. Therefore statement (b) is false.
- (c) We show that if the given statement is true, statement (c) is necessarily true.

We are given that for every  $x$ , there exists  $y$  such that  $P(x, y)$  is true. We must show that there exist  $x$  and  $y$  such that  $P(x, y)$  is true. Since the domain of discourse is nonempty, we may choose a value for  $x$ . For this chosen  $x$ , there exists  $y$  such that  $P(x, y)$  is true. We have found at least one value for  $x$  and at least one value for  $y$  that make  $P(x, y)$  true. Therefore  $\exists x \exists y P(x, y)$  is true.

## Summary of Problem-Solving Techniques

- When dealing with quantified statements, it is sometimes useful to write out the statements in words. For example, in this problem, it helped to write out exactly what  $\forall x \exists y P(x, y)$  means. Take time to let the words sink in.
- If you have trouble finding examples, look at existing examples (e.g., examples in this book). To solve problems (a) and (b), we could have used the statement in Example 1.6.6. Sometimes, an existing example can be modified to solve a given problem.

## Exercises

1. Show that the statement in Example 1.6.6 solves problems (a) and (b) in this Problem-Solving Corner.
2. Could examples in Section 1.6 other than Example 1.6.6 have been used to solve problems (a) and (b) in this Problem-Solving Corner?

## Chapter 1 Notes

General references on discrete mathematics are [Graham, 1994; Liu, 1985; Tucker]. [Knuth, 1997, 1998a, 1998b] is the classic reference for much of this material.

[Halmos; Lipschutz; and Stoll] are recommended to the reader wanting to study set theory in more detail.

[Barker; Copi; Edgar] are introductory logic textbooks. A more advanced treatment is found in [Davis]. The first chapter of the geometry book by [Jacobs] is devoted to basic logic. For a history of logic, see [Kline]. The role of logic in reasoning about computer programs is discussed by [Gries].

## Chapter 1 Review

### Section 1.1

1. Set: any collection of objects
2. Notation for sets:  $\{x \mid x \text{ has property } P\}$
3.  $|X|$ , the cardinality of  $X$ : the number of elements in the set  $X$
4.  $x \in X$ :  $x$  is an element of the set  $X$
5.  $x \notin X$ :  $x$  is not an element of the set  $X$
6. Empty set:  $\emptyset$  or  $\{\}$
7.  $X = Y$ , where  $X$  and  $Y$  are sets:  $X$  and  $Y$  have the same elements
8.  $X \subseteq Y$ ,  $X$  is a subset of  $Y$ : every element in  $X$  is also in  $Y$
9.  $X \subset Y$ ,  $X$  is a proper subset of  $Y$ :  $X \subseteq Y$  and  $X \neq Y$
10.  $\mathcal{P}(X)$ , the power set of  $X$ : set of all subsets of  $X$
11.  $|\mathcal{P}(X)| = 2^{|X|}$

12.  $X \cup Y$ ,  $X$  union  $Y$ : set of elements in  $X$  or  $Y$  or both
13. Union of a family  $\mathcal{S}$  of sets:  $\cup \mathcal{S} = \{x \mid x \in X \text{ for some } X \in \mathcal{S}\}$
14.  $X \cap Y$ ,  $X$  intersect  $Y$ : set of elements in  $X$  and  $Y$
15. Intersection of a family  $\mathcal{S}$  of sets:  $\cap \mathcal{S} = \{x \mid x \in X \text{ for all } X \in \mathcal{S}\}$
16. Disjoint sets  $X$  and  $Y$ :  $X \cap Y = \emptyset$
17. Pairwise disjoint family of sets
18.  $X - Y$ , difference of  $X$  and  $Y$ , relative complement: set of elements in  $X$  but not in  $Y$
19. Universal set, universe
20.  $\overline{X}$ , complement of  $X$ :  $U - X$ , where  $U$  is a universal set
21. Venn diagram

22. Properties of sets (see Theorem 1.1.22)  
 23. De Morgan's laws for sets:  $\overline{(A \cup B)} = \overline{A} \cap \overline{B}$ ,  $\overline{(A \cap B)} = \overline{A} \cup \overline{B}$   
 24. Partition of  $X$ : a collection  $\mathcal{S}$  of nonempty subsets of  $X$  such that every element in  $X$  belongs to exactly one member of  $\mathcal{S}$   
 25. Ordered pair:  $(x, y)$   
 26. Cartesian product of  $X$  and  $Y$ :  $X \times Y = \{(x, y) \mid x \in X, y \in Y\}$   
 27. Cartesian product of  $X_1, X_2, \dots, X_n$ :

$$X_1 \times X_2 \times \cdots \times X_n = \{(a_1, a_2, \dots, a_n) \mid a_i \in X_i\}$$

## Section 1.2

28. Logic  
 29. Proposition  
 30. Conjunction:  $p$  and  $q$ ,  $p \wedge q$   
 31. Disjunction:  $p$  or  $q$ ,  $p \vee q$   
 32. Negation: not  $p$ ,  $\neg p$   
 33. Truth table  
 34. Exclusive-or of propositions  $p, q$ :  $p$  or  $q$ , but not both

## Section 1.3

35. Conditional proposition: if  $p$ , then  $q$ ;  $p \rightarrow q$   
 36. Hypothesis  
 37. Conclusion  
 38. Necessary condition  
 39. Sufficient condition  
 40. Converse of  $p \rightarrow q$ :  $q \rightarrow p$   
 41. Biconditional proposition:  $p$  if and only if  $q$ ,  $p \leftrightarrow q$   
 42. Logical equivalence:  $P \equiv Q$   
 43. De Morgan's laws for logic:  $\neg(p \vee q) \equiv \neg p \wedge \neg q$ ,  $\neg(p \wedge q) \equiv \neg p \vee \neg q$   
 44. Contrapositive of  $p \rightarrow q$ :  $\neg q \rightarrow \neg p$

## Section 1.4

45. Deductive reasoning  
 46. Hypothesis  
 47. Premises  
 48. Conclusion  
 49. Argument  
 50. Valid argument  
 51. Invalid argument  
 52. Rules of inference for propositions: modus ponens, modus tollens, addition, simplification, conjunction, hypothetical syllogism, disjunctive syllogism

## Section 1.5

53. Propositional function  
 54. Domain of discourse

55. Universal quantifier  
 56. Universally quantified statement  
 57. Counterexample  
 58. Existential quantifier  
 59. Existentially quantified statement  
 60. Generalized De Morgan's laws for logic:  
 $\neg(\forall x P(x))$  and  $\exists x \neg P(x)$  have the same truth values.  
 $\neg(\exists x P(x))$  and  $\forall x \neg P(x)$  have the same truth values.

61. To prove that the universally quantified statement  $\forall x P(x)$  is true, show that for every  $x$  in the domain of discourse, the proposition  $P(x)$  is true.  
 62. To prove that the existentially quantified statement  $\exists x P(x)$  is true, find one value of  $x$  in the domain of discourse for which  $P(x)$  is true.  
 63. To prove that the universally quantified statement  $\forall x P(x)$  is false, find one value of  $x$  (a counterexample) in the domain of discourse for which  $P(x)$  is false.  
 64. To prove that the existentially quantified statement  $\exists x P(x)$  is false, show that for every  $x$  in the domain of discourse, the proposition  $P(x)$  is false.  
 65. Rules of inference for quantified statements: universal instantiation, universal generalization, existential instantiation, existential generalization

## Section 1.6

66. To prove that  $\forall x \forall y P(x, y)$  is true, show that  $P(x, y)$  is true for all values of  $x \in X$  and  $y \in Y$ , where the domain of discourse is  $X \times Y$ .  
 67. To prove that  $\forall x \exists y P(x, y)$  is true, show that for all  $x \in X$ , there is at least one  $y \in Y$  such that  $P(x, y)$  is true, where the domain of discourse is  $X \times Y$ .  
 68. To prove that  $\exists x \forall y P(x, y)$  is true, show that for at least one  $x \in X$ ,  $P(x, y)$  is true for every  $y \in Y$ , where the domain of discourse is  $X \times Y$ .  
 69. To prove that  $\exists x \exists y P(x, y)$  is true, find one value of  $x \in X$  and one value of  $y \in Y$  that make  $P(x, y)$  true, where the domain of discourse is  $X \times Y$ .  
 70. To prove that  $\forall x \forall y P(x, y)$  is false, find one value of  $x \in X$  and one value of  $y \in Y$  that make  $P(x, y)$  false, where the domain of discourse is  $X \times Y$ .  
 71. To prove that  $\forall x \exists y P(x, y)$  is false, show that for at least one  $x \in X$ ,  $P(x, y)$  is false for every  $y \in Y$ , where the domain of discourse is  $X \times Y$ .  
 72. To prove that  $\exists x \forall y P(x, y)$  is false, show that for all  $x \in X$ , there is at least one  $y \in Y$  such that  $P(x, y)$  is false, where the domain of discourse is  $X \times Y$ .  
 73. To prove that  $\exists x \exists y P(x, y)$  is false, show that  $P(x, y)$  is false for all values of  $x \in X$  and  $y \in Y$ , where the domain of discourse is  $X \times Y$ .  
 74. To negate an expression with nested quantifiers, use the generalized De Morgan's laws for logic.  
 75. The logic game

## Chapter 1 Self-Test

1. If  $A = \{1, 3, 4, 5, 6, 7\}$ ,  $B = \{x \mid x \text{ is an even integer}\}$ ,  $C = \{2, 3, 4, 5, 6\}$ , find  $(A \cap B) - C$ .
2. If  $p$ ,  $q$ , and  $r$  are true, find the truth value of the proposition  $(p \vee q) \wedge \neg((\neg p \wedge r) \vee q)$ .
3. Restate the proposition “A necessary condition for Leah to get an A in discrete mathematics is to study hard” in the form of a conditional proposition.
4. Write the converse and contrapositive of the proposition of Exercise 3.
5. If  $A \cup B = B$ , what relation must hold between  $A$  and  $B$ ?
6. Are the sets  $\{3, 2, 2\}$ ,  $\{x \mid x \text{ is an integer and } 1 < x \leq 3\}$  equal? Explain.
7. Determine whether the following argument is valid.

$$\begin{array}{c} p \rightarrow q \vee r \\ p \vee \neg q \\ r \vee q \\ \hline \therefore q \end{array}$$

8. If  $p$  is true and  $q$  and  $r$  are false, find the truth value of the proposition  $(p \vee q) \rightarrow \neg r$ .
9. Write the following argument symbolically and determine whether it is valid. If the Skyscrapers win, I'll eat my hat. If I eat my hat, I'll be quite full. Therefore, if I'm quite full, the Skyscrapers won.

10. Is the statement

The team won the 2006 National Basketball Association championship

a proposition? Explain.

11. Is the statement of Exercise 10 a propositional function? Explain.
12. Let  $K(x, y)$  be the propositional function “ $x$  knows  $y$ .” The domain of discourse is the Cartesian product of the set of students taking discrete math with itself (i.e., both  $x$  and  $y$  take on values in the set of students taking discrete math). Represent the assertion “someone does not know anyone” symbolically.
13. Write the negation of the assertion of Exercise 12 symbolically and in words.

14. If  $A = \{a, b, c\}$ , how many elements are in  $\mathcal{P}(A) \times A$ ?
15. Write the truth table of the proposition  $\neg(p \wedge q) \vee (p \vee \neg r)$ .
16. Formulate the proposition  $p \wedge (\neg q \vee r)$  in words using

$p$ : I take hotel management.  
 $q$ : I take recreation supervision.  
 $r$ : I take popular culture.

17. Assume that  $a$ ,  $b$ , and  $c$  are real numbers. Represent the statement

$$a < b \text{ or } (b < c \text{ and } a \geq c)$$

symbolically, letting

$$p: a < b, \quad q: b < c, \quad r: a \geq c.$$

Let  $P(n)$  be the statement  $n$  and  $n + 2$  are prime. In Exercises 18 and 19, write the statement in words and tell whether it is true or false.

18.  $\forall n P(n)$
19.  $\exists n P(n)$
20. Which rule of inference is used in the following argument? If the Skyscrapers win, I'll eat my hat. If I eat my hat, I'll be quite full. Therefore, if the Skyscrapers win, I'll be quite full.
21. Give an argument using rules of inference to show that the conclusion follows from the hypotheses.

Hypotheses: If the Council approves the funds, then New Atlantic will get the Olympic Games. If New Atlantic gets the Olympic Games, then New Atlantic will build a new stadium. New Atlantic does not build a new stadium. Conclusion: The Council does not approve the funds, or the Olympic Games are canceled.

22. Determine whether the statement  $\forall x \exists y (x = y^3)$  is true or false. The domain of discourse is  $\mathbf{R} \times \mathbf{R}$ . Explain your answer. Explain, in words, the meaning of the statement.
23. Use the generalized De Morgan's laws for logic to write the negation of  $\forall x \exists y \forall z P(x, y, z)$ .
24. Represent the statement

$$\text{If } (a \geq c \text{ or } b < c), \text{ then } b \geq c$$

symbolically using the definitions of Exercise 17.

## Chapter 1 Computer Exercises

In Exercises 1–6, assume that a set  $X$  of  $n$  elements is represented as an array  $A$  of size at least  $n + 1$ . The elements of  $X$  are listed consecutively in  $A$  starting in the first position and terminating with 0. Assume further that no set contains 0.

1. Write a program to represent the sets  $X \cup Y$ ,  $X \cap Y$ ,  $X - Y$ , and  $X \Delta Y$ , given the arrays representing  $X$  and  $Y$ . (The symmetric difference is denoted  $\Delta$ .)

2. Write a program to determine whether  $X \subseteq Y$ , given arrays representing  $X$  and  $Y$ .
3. Write a program to determine whether  $X = Y$ , given arrays representing  $X$  and  $Y$ .
4. Assuming a universe represented as an array, write a program to represent the set  $\bar{X}$ , given the array representing  $X$ .
5. Given an element  $E$  and the array  $A$  that represents  $X$ , write a program that determines whether  $E \in X$ .
6. Given the array representing  $X$ , write a program that lists all subsets of  $X$ .
7. Write a program that reads a logical expression in  $p$  and  $q$  and prints the truth table of the expression.
8. Write a program that reads a logical expression in  $p$ ,  $q$ , and  $r$  and prints the truth table of the expression.
9. Write a program that tests whether two logical expressions in  $p$  and  $q$  are logically equivalent.
10. Write a program that tests whether two logical expressions in  $p$ ,  $q$ , and  $r$  are logically equivalent.



## Chapter 2

# PROOFS

- 2.1** Mathematical Systems, Direct Proofs, and Counterexamples
- 2.2** More Methods of Proof
- †2.3** Resolution Proofs
- 2.4** Mathematical Induction
- 2.5** Strong Form of Induction and the Well-Ordering Property

### Go Online

For more on proofs, see  
[goo.gl/gHgyey](http://goo.gl/gHgyey)

This chapter uses the logic in Chapter 1 to discuss proofs. Logical methods are used in mathematics to prove theorems and in computer science to prove that programs do what they are alleged to do. Suppose, for example, that a student is assigned a program to compute shortest paths between cities. The program must be able to accept as input an arbitrary number of cities and the distances between cities directly connected by roads and produce as output the shortest paths (routes) between each distinct pair of cities. After the student writes the program, it is easy to test it for a small number of cities. Using paper and pencil, the student could simply list all possible paths between pairs of cities and find the shortest paths. This brute-force solution could then be compared with the output of the program. However, for a large number of cities, the brute-force technique would take too long. How can the student be sure that the program works properly for large input—almost surely the kind of input on which the instructor will test the program? The student will have to use logic to prove that the program is correct. The proof might be informal or formal using the techniques presented in this chapter, but a proof will be required.

After introducing some context and terminology in Section 2.1, we devote the remainder of this chapter to various proof techniques. Sections 2.1 and 2.2 introduce several proof techniques that build directly on the material in Chapter 1. Resolution, the topic of Section 2.3, is a special proof technique that can be automated. Sections 2.4 and 2.5 are concerned with mathematical induction, a proof technique especially useful in discrete mathematics and computer science.

---

<sup>†</sup>This section can be omitted without loss of continuity.

## 2.1 Mathematical Systems, Direct Proofs, and Counterexamples

A **mathematical system** consists of **axioms**, **definitions**, and **undefined terms**. Axioms are assumed to be true. Definitions are used to create new concepts in terms of existing ones. Some terms are not explicitly defined but rather are implicitly defined by the axioms. Within a mathematical system we can derive theorems. A **theorem** is a proposition that has been proved to be true. Special kinds of theorems are referred to as lemmas and corollaries. A **lemma** is a theorem that is usually not too interesting in its own right but is useful in proving another theorem. A **corollary** is a theorem that follows easily from another theorem.

An argument that establishes the truth of a theorem is called a **proof**. Logic is a tool for the analysis of proofs. In this section and the next, we introduce some general methods of proof. In Sections 2.3–2.5, we discuss resolution and mathematical induction, which are special proof techniques. We begin by giving some examples of mathematical systems.

### Example 2.1.1

Euclidean geometry furnishes an example of a mathematical system. Among the axioms are

- Given two distinct points, there is exactly one line that contains them.
- Given a line and a point not on the line, there is exactly one line parallel to the line through the point.

The terms *point* and *line* are undefined terms that are implicitly defined by the axioms that describe their properties.

Among the definitions are

- Two triangles are *congruent* if their vertices can be paired so that the corresponding sides and corresponding angles are equal.
- Two angles are *supplementary* if the sum of their measures is  $180^\circ$ .

### Example 2.1.2

The real numbers furnish another example of a mathematical system. Among the axioms are

- For all real numbers  $x$  and  $y$ ,  $xy = yx$ .
- There is a subset  $\mathbf{P}$  of real numbers satisfying
  - (a) If  $x$  and  $y$  are in  $\mathbf{P}$ , then  $x + y$  and  $xy$  are in  $\mathbf{P}$ .
  - (b) If  $x$  is a real number, then exactly one of the following statements is true:

$$x \text{ is in } \mathbf{P}, \quad x = 0, \quad -x \text{ is in } \mathbf{P}.$$

Multiplication is implicitly defined by the first axiom and others that describe the properties multiplication is assumed to have.

Among the definitions are

- The elements in  $\mathbf{P}$  (of the preceding axiom) are called *positive real numbers*.
- The *absolute value*  $|x|$  of a real number  $x$  is defined to be  $x$  if  $x$  is positive or 0 and  $-x$  otherwise.

We give several examples of theorems, corollaries, and lemmas in Euclidean geometry and in the system of real numbers.

**Example 2.1.3**

Examples of theorems in Euclidean geometry are

- If two sides of a triangle are equal, then the angles opposite them are equal.
- If the diagonals of a quadrilateral bisect each other, then the quadrilateral is a parallelogram.

**Example 2.1.4**

An example of a corollary in Euclidean geometry is

- If a triangle is equilateral, then it is equiangular.

This corollary follows immediately from the first theorem of Example 2.1.3.

**Example 2.1.5**

Examples of theorems about real numbers are

- $x \cdot 0 = 0$  for every real number  $x$ .
- For all real numbers  $x$ ,  $y$ , and  $z$ , if  $x \leq y$  and  $y \leq z$ , then  $x \leq z$ .

**Example 2.1.6**

An example of a lemma about real numbers is

- If  $n$  is a positive integer, then either  $n - 1$  is a positive integer or  $n - 1 = 0$ .

Surely this result is not that interesting in its own right, but it can be used to prove other results.

**Direct Proofs**

Theorems are often of the form

For all  $x_1, x_2, \dots, x_n$ , if  $p(x_1, x_2, \dots, x_n)$ , then  $q(x_1, x_2, \dots, x_n)$ .

This universally quantified statement is true provided that the conditional proposition

$$\text{if } p(x_1, x_2, \dots, x_n), \text{ then } q(x_1, x_2, \dots, x_n) \quad (2.1.1)$$

is true for all  $x_1, x_2, \dots, x_n$  in the domain of discourse. To prove (2.1.1), we assume that  $x_1, x_2, \dots, x_n$  are arbitrary members of the domain of discourse. If  $p(x_1, x_2, \dots, x_n)$  is false, by Definition 1.3.3, (2.1.1) is vacuously true; thus, we need only consider the case that  $p(x_1, x_2, \dots, x_n)$  is true. A **direct proof** assumes that  $p(x_1, x_2, \dots, x_n)$  is true and then, using  $p(x_1, x_2, \dots, x_n)$  as well as other axioms, definitions, previously derived theorems, and rules of inference, shows directly that  $q(x_1, x_2, \dots, x_n)$  is true.

Everyone “knows” what an even or odd integer is, but the following definition makes these terms precise and provides a formal way to use the terms “even integer” and “odd integer” in proofs.

**Definition 2.1.7** ► An integer  $n$  is *even* if there exists an integer  $k$  such that  $n = 2k$ . An integer  $n$  is *odd* if there exists an integer  $k$  such that  $n = 2k + 1$ .

**Example 2.1.8**

The integer  $n = 12$  is even because there exists an integer  $k$  (namely  $k = 6$ ) such that  $n = 2k$ ; that is,  $12 = 2 \cdot 6$ .

**Example 2.1.9**

The integer  $n = -21$  is odd because there exists an integer  $k$  (namely  $k = -11$ ) such that  $n = 2k + 1$ ; that is,  $-21 = 2(-11) + 1$ .

**Example 2.1.10**

Give a direct proof of the following statement. For all integers  $m$  and  $n$ , if  $m$  is odd and  $n$  is even, then  $m + n$  is odd.

**SOLUTION Discussion** In a direct proof, we assume the hypotheses and derive the conclusion. A good start is achieved by writing out the hypotheses and conclusion so that we are clear where we start and where we are headed. In the case at hand, we have

$m$  is odd and  $n$  is even. (Hypotheses)

...

$m + n$  is odd. (Conclusion)

The gap ( $\dots$ ) represents the part of the proof to be completed that leads from the hypotheses to the conclusion.

We can begin to fill in the gap by using the definitions of “odd” and “even” to obtain

$m$  is odd and  $n$  is even. (Hypotheses)

There exists an integer, say  $k_1$ , such that  $m = 2k_1 + 1$ . (Because  $m$  is odd)

There exists an integer, say  $k_2$ , such that  $n = 2k_2$ . (Because  $n$  is even)

...

$m + n$  is odd. (Conclusion)

(Notice that we *cannot* assume that  $k_1 = k_2$ . For example if  $m = 15$  and  $n = 4$ , then  $k_1 = 7$  and  $k_2 = 2$ . That  $k_1$  is not necessarily equal to  $k_2$  is the reason that we *must* denote the two integers with different symbols.)

The missing part of our proof is the argument to show that  $m + n$  is odd. How can we reach this conclusion? We can use the definition of “odd” again if we can show that  $m + n$  is equal to

$$2 \times \text{some integer} + 1. \quad (2.1.2)$$

We already know that  $m = 2k_1 + 1$  and  $n = 2k_2$ . How can we use these facts to reach our goal (2.1.2)? Since the goal involves  $m + n$ , we can add the equations  $m = 2k_1 + 1$  and  $n = 2k_2$  to obtain a fact about  $m + n$ , namely,

$$m + n = (2k_1 + 1) + 2k_2.$$

Now this expression is supposed to be of the form (2.1.2). We can use a little algebra to show that it is of the desired form:

$$m + n = (2k_1 + 1) + 2k_2 = 2(k_1 + k_2) + 1.$$

We have our proof.

**Proof** Let  $m$  and  $n$  be arbitrary integers, and suppose that  $m$  is odd and  $n$  is even. We prove that  $m + n$  is odd. By definition, since  $m$  is odd, there exists an integer  $k_1$  such that  $m = 2k_1 + 1$ . Also, by definition, since  $n$  is even, there exists an integer  $k_2$  such that  $n = 2k_2$ . Now the sum is

$$m + n = (2k_1 + 1) + (2k_2) = 2(k_1 + k_2) + 1.$$

Thus, there exists an integer  $k$  (namely  $k = k_1 + k_2$ ) such that  $m + n = 2k + 1$ . Therefore,  $m + n$  is odd. 

### Example 2.1.11

Give a direct proof of the following statement. For all sets  $X$ ,  $Y$ , and  $Z$ ,  $X \cap (Y - Z) = (X \cap Y) - (X \cap Z)$ .

**SOLUTION Discussion** The outline of the proof is

$X, Y$ , and  $Z$  are sets. (Hypothesis)

...

$X \cap (Y - Z) = (X \cap Y) - (X \cap Z)$  (Conclusion)

The conclusion asserts that the two sets  $X \cap (Y - Z)$  and  $(X \cap Y) - (X \cap Z)$  are equal. Recall (see Section 1.1) that to prove from the definition of set equality that these sets are equal, we must show that for all  $x$ ,

$$\text{if } x \in X \cap (Y - Z), \text{ then } x \in (X \cap Y) - (X \cap Z) \quad (2.1.3)$$

and

$$\text{if } x \in (X \cap Y) - (X \cap Z), \text{ then } x \in X \cap (Y - Z). \quad (2.1.4)$$

Thus our proof outline becomes

$X, Y$ , and  $Z$  are sets. (Hypothesis)

If  $x \in X \cap (Y - Z)$ , then  $x \in (X \cap Y) - (X \cap Z)$ .

If  $x \in (X \cap Y) - (X \cap Z)$ , then  $x \in X \cap (Y - Z)$ .

$X \cap (Y - Z) = (X \cap Y) - (X \cap Z)$  (Conclusion)

We should be able to use the definitions of intersection ( $\cap$ ) and set difference ( $-$ ) to complete the proof.

To prove (2.1.3), we begin by assuming that (the arbitrary element)  $x$  is in  $X \cap (Y - Z)$ . Because this latter set is an intersection, we immediately deduce that  $x \in X$  and  $x \in Y - Z$ . The proof proceeds in this way. As one constructs the proof, it is essential to keep the goal in mind:  $x \in (X \cap Y) - (X \cap Z)$ . To help guide the construction of the proof, it may be helpful to translate the goal using the definition of set difference:  $x \in (X \cap Y) - (X \cap Z)$  means  $x \in X \cap Y$  and  $x \notin X \cap Z$ .

**Proof** Let  $X, Y$ , and  $Z$  be arbitrary sets. We prove

$$X \cap (Y - Z) = (X \cap Y) - (X \cap Z)$$

by proving (2.1.3) and (2.1.4).

To prove equation (2.1.3), let  $x \in X \cap (Y - Z)$ . By the definition of intersection,  $x \in X$  and  $x \in Y - Z$ . By the definition of set difference, since  $x \in Y - Z$ ,  $x \in Y$  and  $x \notin Z$ . By the definition of intersection, since  $x \in X$  and  $x \in Y$ ,  $x \in X \cap Y$ . Again by the definition of intersection, since  $x \notin Z$ ,  $x \notin X \cap Z$ . By the definition of set difference, since  $x \in X \cap Y$ , but  $x \notin X \cap Z$ ,  $x \in (X \cap Y) - (X \cap Z)$ . We have proved equation (2.1.3).

To prove equation (2.1.4), let  $x \in (X \cap Y) - (X \cap Z)$ . By the definition of set difference,  $x \in X \cap Y$  and  $x \notin X \cap Z$ . By the definition of intersection, since  $x \in X \cap Y$ ,  $x \in X$  and  $x \in Y$ . Again, by the definition of intersection, since  $x \notin X \cap Z$  and  $x \in X$ ,  $x \notin Z$ . By the definition of set difference, since  $x \in Y$  and  $x \notin Z$ ,  $x \in Y - Z$ . Finally, by the definition of intersection, since  $x \in X$  and  $x \in Y - Z$ ,  $x \in X \cap (Y - Z)$ . We have proved equation (2.1.4).

Since we have proved both equations (2.1.3) and (2.1.4), it follows that

$$X \cap (Y - Z) = (X \cap Y) - (X \cap Z).$$



Our next example shows that in constructing a proof, we may find that we need some auxiliary results, at which point we pause, go off and prove these auxiliary results, and then return to the main proof. We call the proofs of auxiliary results **subproofs**. (For those familiar with programming, a subproof is similar to a subroutine.)

**Example 2.1.12**

If  $a$  and  $b$  are real numbers, we define  $\min\{a, b\}$  to be the *minimum* of  $a$  and  $b$  or the common value if they are equal. More precisely,

$$\min\{a, b\} = \begin{cases} a & \text{if } a < b \\ a & \text{if } a = b \\ b & \text{if } b < a. \end{cases}$$

Give a direct proof of the following statement. For all real numbers  $d, d_1, d_2, x$ ,

$$\text{if } d = \min\{d_1, d_2\} \text{ and } x \leq d, \text{ then } x \leq d_1 \text{ and } x \leq d_2.$$

**SOLUTION Discussion** The outline of the proof is

$$d = \min\{d_1, d_2\} \text{ and } x \leq d \text{ (Hypotheses)}$$

...

$$x \leq d_1 \text{ and } x \leq d_2 \text{ (Conclusion)}$$

To help understand what is being asserted, let us look at a specific example. As we have remarked previously, when we are asked to prove a universally quantified statement, a specific example does not *prove* the statement. It may, however, help us to *understand* the statement.

Let us set  $d_1 = 2$  and  $d_2 = 4$ . Then  $d = \min\{d_1, d_2\} = 2$ . The statement to be proved says that if  $x \leq d (= 2)$ , then  $x \leq d_1 (= 2)$  and  $x \leq d_2 (= 4)$ . Why is this true in general? The minimum  $d$  of two numbers,  $d_1$  and  $d_2$ , is equal to one of the two numbers (namely, the smallest) and less than or equal to the other one (namely, the largest)—in symbols,  $d \leq d_1$  and  $d \leq d_2$ . If  $x \leq d$ , then from  $x \leq d$  and  $d \leq d_1$ , we may deduce  $x \leq d_1$ . Similarly, from  $x \leq d$  and  $d \leq d_2$ , we may deduce  $x \leq d_2$ . Thus the outline of our proof becomes

$$d = \min\{d_1, d_2\} \text{ and } x \leq d \text{ (Hypotheses)}$$

Subproof: Show that  $d \leq d_1$  and  $d \leq d_2$ .

From  $x \leq d$  and  $d \leq d_1$ , deduce  $x \leq d_1$ .

From  $x \leq d$  and  $d \leq d_2$ , deduce  $x \leq d_2$ .

$$x \leq d_1 \text{ and } x \leq d_2 \text{ (Conclusion—uses the conjunction inference rule)}$$

At this point, the only part of the proof that is missing is the subproof to show that  $d \leq d_1$  and  $d \leq d_2$ . Let us look at the definition of “minimum.” If  $d_1 \leq d_2$ , then  $d = \min\{d_1, d_2\} = d_1$  and  $d = d_1 \leq d_2$ . If  $d_2 < d_1$ , then  $d = \min\{d_1, d_2\} = d_2$  and  $d = d_2 < d_1$ . In either case,  $d \leq d_1$  and  $d \leq d_2$ .

**Proof** Let  $d, d_1, d_2$ , and  $x$  be arbitrary real numbers, and suppose that

$$d = \min\{d_1, d_2\} \text{ and } x \leq d.$$

We prove that  $x \leq d_1$  and  $x \leq d_2$ .

We first show that  $d \leq d_1$  and  $d \leq d_2$ . From the definition of “minimum,” if  $d_1 \leq d_2$ , then  $d = \min\{d_1, d_2\} = d_1$  and  $d = d_1 \leq d_2$ . If  $d_2 < d_1$ , then  $d = \min\{d_1, d_2\} = d_2$  and  $d = d_2 < d_1$ . In either case,  $d \leq d_1$  and  $d \leq d_2$ . From  $x \leq d$  and  $d \leq d_1$ , it follows that  $x \leq d_1$  from a previous theorem (the second theorem of Example 2.1.5). From  $x \leq d$  and  $d \leq d_2$ , we may derive  $x \leq d_2$  from the same previous theorem. Therefore,  $x \leq d_1$  and  $x \leq d_2$ .  

**Example 2.1.13**

There are frequently many different ways to prove a statement. We illustrate by giving two proofs of the statement

$$X \cup (Y - X) = X \cup Y \quad \text{for all sets } X \text{ and } Y.$$

**SOLUTION Discussion** We first give a direct proof like the proof in Example 2.1.11. We show that for all  $x$ , if  $x \in X \cup (Y - X)$ , then  $x \in X \cup Y$ , and if  $x \in X \cup Y$ , then  $x \in X \cup (Y - X)$ .

Our second proof uses Theorem 1.1.22, which gives laws of sets. The idea is to begin with  $X \cup (Y - X)$  and use the laws of sets, which here we think of as rules to manipulate set equations, to obtain  $X \cup Y$ .

**Proof** [First proof] We show that for all  $x$ , if  $x \in X \cup (Y - X)$ , then  $x \in X \cup Y$ , and if  $x \in X \cup Y$ , then  $x \in X \cup (Y - X)$ .

Let  $x \in X \cup (Y - X)$ . Then  $x \in X$  or  $x \in Y - X$ . If  $x \in X$ , then  $x \in X \cup Y$ . If  $x \in Y - X$ , then  $x \in Y$ , so again  $x \in X \cup Y$ . In either case,  $x \in X \cup Y$ .

Let  $x \in X \cup Y$ . Then  $x \in X$  or  $x \in Y$ . If  $x \in X$ , then  $x \in X \cup (Y - X)$ . If  $x \notin X$ , then  $x \in Y$ . In this case,  $x \in Y - X$ . Therefore,  $x \in X \cup (Y - X)$ . In either case,  $x \in X \cup (Y - X)$ . The proof is complete. ◀

**Proof** [Second proof] We use Theorem 1.1.22, which gives laws of sets, and the fact that  $Y - X = Y \cap \bar{X}$ , which follows immediately from the definition of set difference. Letting  $U$  denote the universal set, we obtain

$$\begin{aligned} X \cup (Y - X) &= X \cup (Y \cap \bar{X}) && [Y - X = Y \cap \bar{X}] \\ &= (X \cup Y) \cap (X \cup \bar{X}) && [\text{Distributive law; Theorem 1.1.22, part (c)}] \\ &= (X \cup Y) \cap U && [\text{Complement law; Theorem 1.1.22, part (e)}] \\ &= X \cup Y && [\text{Identity law; Theorem 1.1.22, part (d)}]. \end{aligned} \quad \blacktriangleleft \quad \blacktriangleright$$

### Disproving a Universally Quantified Statement

Recall (see Section 1.5) that to *disprove*  $\forall x P(x)$  we simply need to find one member  $x$  in the domain of discourse that makes  $P(x)$  false. Such a value for  $x$  is called a *counterexample*.

#### Example 2.1.14

The statement  $\forall n \in \mathbf{Z}^+ (2^n + 1 \text{ is prime})$  is false. A counterexample is  $n = 3$  since  $2^3 + 1 = 9$ , which is not prime. ◀

#### Example 2.1.15

If the statement

$$(A \cap B) \cup C = A \cap (B \cup C), \quad \text{for all sets } A, B, \text{ and } C$$

is true, prove it; otherwise, give a counterexample.

**SOLUTION** Let us begin by trying to prove the statement. We will first try to show that if  $x \in (A \cap B) \cup C$ , then  $x \in A \cap (B \cup C)$ . If  $x \in (A \cap B) \cup C$ , then

$$x \in A \cap B \quad \text{or} \quad x \in C. \tag{2.1.5}$$

We have to show that  $x \in A \cap (B \cup C)$ , that is,

$$x \in A \quad \text{and} \quad x \in B \cup C. \tag{2.1.6}$$

Statement (2.1.5) is true if  $x$  is in  $C$ , and statement (2.1.6) is false if  $x \notin A$ . Thus the given statement is false; there is no direct proof (or any other proof!). If we choose sets  $A$  and  $C$  so that there is an element that is in  $C$ , but not in  $A$ , we will have a counterexample.

Let  $A = \{1, 2, 3\}$ ,  $B = \{2, 3, 4\}$ , and  $C = \{3, 4, 5\}$  so that there is an element that is in  $C$ , but not in  $A$ . Then  $(A \cap B) \cup C = \{2, 3, 4, 5\}$ ,  $A \cap (B \cup C) = \{2, 3\}$ , and  $(A \cap B) \cup C \neq A \cap (B \cup C)$ . Thus  $A$ ,  $B$ , and  $C$  provide a counterexample that shows that the given statement is false. ◀

## 2.1 Problem-Solving Tips

- To construct a direct proof of a universally quantified statement, first write down the hypotheses (so you know what you are assuming), and then write down the conclusion (so you know what you must prove). The conclusion is what you will work toward—something like the answer in the back of the book to an exercise, except here it is essential to know the goal before proceeding. You must now give an argument that begins with the hypotheses and ends with the conclusion. To construct the argument, remind yourself what you know about the terms (e.g., “even,” “odd”), symbols (e.g.,  $X \cap Y$ ,  $\min\{d_1, d_2\}$ ), and so on. Look at relevant definitions and related results. For example, if a particular hypothesis refers to an even integer  $n$ , you know that  $n$  is of the form  $2k$  for some integer  $k$ . If you are to prove that two sets  $X$  and  $Y$  are equal from the definition of set equality, you know you must show that for every  $x$ , if  $x \in X$  then  $x \in Y$ , and if  $x \in Y$  then  $x \in X$ .
- To understand what is to be proved, look at some specific values in the domain of discourse. When we are asked to prove a universally quantified statement, showing that the statement is true for specific values does not *prove* the statement; it may, however, help to *understand* the statement.
- To *disprove* a universally quantified statement, find *one element* in the domain of discourse, called a *counterexample*, that makes the propositional function false. Here, your proof consists of presenting the counterexample together with justification that the propositional function is indeed false for your counterexample.
- When you write up your proof, begin by writing out the statement to be proved. Indicate clearly where your proof begins (e.g., by beginning a new paragraph or by writing “Proof.”). Use complete sentences, which may include symbols. For example, it is perfectly acceptable to write: Thus  $x \in X$ . In words, this is the complete sentence: Thus  $x$  is in  $X$ . End a direct proof by clearly stating the conclusion, and, perhaps, giving a reason to justify the conclusion. For example, Example 2.1.10 ends with:

Thus, there exists an integer  $k$  (namely  $k = k_1 + k_2$ ) such that  
 $m + n = 2k + 1$ . Therefore,  $m + n$  is odd.

Here the conclusion ( $m + n$  is odd) is clearly stated and justified by the statement  $m + n = 2k + 1$ .

- Alert the reader where you are headed. For example, if you are going to prove that  $X = Y$ , write “We will prove that  $X = Y$ ” before launching into this part of the proof.
- Justify your steps. For example, if you conclude that  $x \in X$  or  $x \in Y$  because it is known that  $x \in X \cup Y$ , write “Since  $x \in X \cup Y$ ,  $x \in X$  or  $x \in Y$ ,” or perhaps even “Since  $x \in X \cup Y$ , by the definition of union  $x \in X$  or  $x \in Y$ ” if, like Richard Nixon, you want to be perfectly clear.
- If you are asked to prove or disprove a universally quantified statement, you can begin by trying to prove it. If you succeed, you are finished—the statement is true and you proved it! If your proof breaks down, look carefully at the point where it fails. The given statement may be false and your failed proof may give insight into how to construct a counterexample (see Example 2.1.15). On the other hand, if you have trouble constructing a counterexample, check where your proposed examples fail. This insight may show why the statement is true and guide construction of a proof.

### Some Common Errors

In Example 2.1.10, we pointed out that it is an error to use the same notation for two possibly distinct quantities. As an example, here is a faulty “proof” that for all  $m$  and  $n$ , if  $m$  and  $n$  are even integers then  $mn$  is a square (i.e.,  $mn = a^2$  for some integer  $a$ ): Since  $m$  and  $n$  are even,  $m = 2k$  and  $n = 2k$ . Now  $mn = (2k)(2k) = (2k)^2$ . If we let  $a = 2k$ , then  $m = a^2$ . The problem is that we *cannot* use  $k$  for two potentially different quantities. If  $m$  and  $n$  are even, all we can conclude is that  $m = 2k_1$  and  $n = 2k_2$  for some integers  $k_1$  and  $k_2$ . The integers  $k_1$  and  $k_2$  need *not* be equal. (In fact, it is false that for all  $m$  and  $n$ , if  $m$  and  $n$  are even integers then  $mn$  is a square. A counterexample is  $m = 2$  and  $n = 4$ .)

Given a universally quantified propositional function, showing that the propositional function is true for specific values in the domain of discourse is *not* a proof that the propositional function is true for all values in the domain of discourse. (Such specific values may, however, *suggest* that the propositional function *might* be true for all values in the domain of discourse.) Example 2.1.10 is to prove that for *all* integers  $m$  and  $n$ ,

$$\text{if } m \text{ is odd and } n \text{ is even, then } m + n \text{ is odd.} \quad (2.1.7)$$

Letting  $m = 11$  and  $n = 4$  and noting that  $m + n = 15$  is odd does *not* constitute a proof that (2.1.7) is true for all integers  $m$  and  $n$ , it merely proves that (2.1.7) is true for the specific values  $m = 11$  and  $n = 4$ .

In constructing a proof, you cannot assume what you are supposed to prove. As an example, consider the *erroneous* “proof” that for all integers  $m$  and  $n$ , if  $m$  and  $m + n$  are even, then  $n$  is even: Let  $m = 2k_1$  and  $n = 2k_2$ . Then  $m + n = 2k_1 + 2k_2$ . Therefore,

$$n = (m + n) - m = (2k_1 + 2k_2) - 2k_1 = 2(k_1 + k_2 - k_1).$$

Thus  $n$  is even. The problem with the preceding “proof” is that we cannot write  $n = 2k_2$  since this is true if and only if  $n$  is even—which is what we are supposed to prove! This error is called **begging the question** or **circular reasoning**. [It is true that if  $m$  and  $m + n$  are even integers, then  $n$  is even (see Exercise 12).]

## 2.1 Review Exercises

1. What is a mathematical system?
2. What is an axiom?
3. What is a definition?
4. What is an undefined term?
5. What is a theorem?
6. What is a proof?
7. What is a lemma?
8. What is a direct proof?
9. What is the formal definition of “even integer”?
10. What is the formal definition of “odd integer”?
11. What is a subproof?
12. How do you disprove a universally quantified statement?

## 2.1 Exercises

1. Give an example (different from those of Example 2.1.1) of an axiom in Euclidean geometry.
2. Give an example (different from those of Example 2.1.2) of an axiom in the system of real numbers.
3. Give an example (different from those of Example 2.1.1) of a definition in Euclidean geometry.
4. Give an example (different from those of Example 2.1.2) of a definition in the system of real numbers.
5. Give an example (different from those of Example 2.1.3) of a theorem in Euclidean geometry.
6. Give an example (different from those of Example 2.1.5) of a theorem in the system of real numbers.
7. Prove that for all integers  $m$  and  $n$ , if  $m$  and  $n$  are even, then  $m + n$  is even.
8. Prove that for all integers  $m$  and  $n$ , if  $m$  and  $n$  are odd, then  $m + n$  is even.

9. Prove that for all integers  $m$  and  $n$ , if  $m$  and  $n$  are even, then  $mn$  is even.
  10. Prove that for all integers  $m$  and  $n$ , if  $m$  and  $n$  are odd, then  $mn$  is odd.
  11. Prove that for all integers  $m$  and  $n$ , if  $m$  is odd and  $n$  is even, then  $mn$  is even.
  12. Prove that for all integers  $m$  and  $n$ , if  $m$  and  $m+n$  are even, then  $n$  is even.
  13. Prove that for all rational numbers  $x$  and  $y$ ,  $x+y$  is rational.
  14. Prove that for all rational numbers  $x$  and  $y$ ,  $xy$  is rational.
  15. Prove that for every rational number  $x$ , if  $x \neq 0$ , then  $1/x$  is rational.
  16. Prove that the product of two integers, one of the form  $3k_1 + 1$  and the other of the form  $3k_2 + 2$ , where  $k_1$  and  $k_2$  are integers, is of the form  $3k_3 + 2$  for some integer  $k_3$ .
  17. Prove that the product of two integers, one of the form  $3k_1 + 2$  and the other of the form  $3k_2 + 2$ , where  $k_1$  and  $k_2$  are integers, is of the form  $3k_3 + 1$  for some integer  $k_3$ .
  18. If  $a$  and  $b$  are real numbers, we define  $\max\{a, b\}$  to be the *maximum* of  $a$  and  $b$  or the common value if they are equal. Prove that for all real numbers  $d, d_1, d_2, x$ ,
 

if  $d = \max\{d_1, d_2\}$  and  $x \geq d$ , then  $x \geq d_1$  and  $x \geq d_2$ .
  19. Justify each step of the following direct proof, which shows that if  $x$  is a real number, then  $x \cdot 0 = 0$ . Assume that the following are previous theorems: If  $a, b$ , and  $c$  are real numbers, then  $b+0 = b$  and  $a(b+c) = ab+ac$ . If  $a+b = a+c$ , then  $b=c$ .
 

**Proof**  $x \cdot 0 + 0 = x \cdot 0 = x \cdot (0 + 0) = x \cdot 0 + x \cdot 0$ ; therefore,  $x \cdot 0 = 0$ . ◀
  20. If  $X$  and  $Y$  are nonempty sets and  $X \times Y = Y \times X$ , what can we conclude about  $X$  and  $Y$ ? Prove your answer.
  21. Prove that  $X \cap Y \subseteq X$  for all sets  $X$  and  $Y$ .
  22. Prove that  $X \subseteq X \cup Y$  for all sets  $X$  and  $Y$ .
  23. Prove that if  $X \subseteq Y$ , then  $X \cup Z \subseteq Y \cup Z$  for all sets  $X, Y$ , and  $Z$ .
  24. Prove that if  $X \subseteq Y$ , then  $X \cap Z \subseteq Y \cap Z$  for all sets  $X, Y$ , and  $Z$ .
  25. Prove that if  $X \subseteq Y$ , then  $Z - Y \subseteq Z - X$  for all sets  $X, Y$ , and  $Z$ .
  26. Prove that if  $X \subseteq Y$ , then  $Y - (Y - X) = X$  for all sets  $X$  and  $Y$ .
  27. Prove that if  $X \cap Y = X \cap Z$  and  $X \cup Y = X \cup Z$ , then  $Y = Z$  for all sets  $X, Y$ , and  $Z$ .
  28. Prove that  $\mathcal{P}(X) \cup \mathcal{P}(Y) \subseteq \mathcal{P}(X \cup Y)$  for all sets  $X$  and  $Y$ .
  29. Prove that  $\mathcal{P}(X \cap Y) = \mathcal{P}(X) \cap \mathcal{P}(Y)$  for all sets  $X$  and  $Y$ .
  30. Prove that if  $\mathcal{P}(X) \subseteq \mathcal{P}(Y)$ , then  $X \subseteq Y$  for all sets  $X$  and  $Y$ .
  31. Disprove that  $\mathcal{P}(X \cup Y) \subseteq \mathcal{P}(X) \cup \mathcal{P}(Y)$  for all sets  $X$  and  $Y$ .
  32. Give a direct proof along the lines of the second proof in Example 2.1.13 of the statement
- $X \cap (Y - Z) = (X \cap Y) - (X \cap Z)$  for all sets  $X, Y$ , and  $Z$ . (In Example 2.1.11, we gave a direct proof of this statement using the definition of set equality.)

In each of Exercises 33–45, if the statement is true, prove it; otherwise, give a counterexample. The sets  $X$ ,  $Y$ , and  $Z$  are subsets of a universal set  $U$ . Assume that the universe for Cartesian products is  $U \times U$ .

33. For all sets  $X$  and  $Y$ , either  $X$  is a subset of  $Y$  or  $Y$  is a subset of  $X$ .
  34.  $X \cup (Y - Z) = (X \cup Y) - (X \cup Z)$  for all sets  $X, Y$ , and  $Z$ .
  35.  $\overline{Y - X} = X \cup \overline{Y}$  for all sets  $X$  and  $Y$ .
  36.  $Y - Z = (X \cup Y) - (X \cup Z)$  for all sets  $X, Y$ , and  $Z$ .
  37.  $X - (Y \cup Z) = (X - Y) \cup Z$  for all sets  $X, Y$ , and  $Z$ .
  38.  $\overline{X - Y} = \overline{Y - X}$  for all sets  $X$  and  $Y$ .
  39.  $\overline{X \cap Y} \subseteq X$  for all sets  $X$  and  $Y$ .
  40.  $(X \cap Y) \cup (Y - X) = Y$  for all sets  $X$  and  $Y$ .
  41.  $X \times (Y \cup Z) = (X \times Y) \cup (X \times Z)$  for all sets  $X, Y$ , and  $Z$ .
  42.  $\overline{X \times Y} = \overline{X} \times \overline{Y}$  for all sets  $X$  and  $Y$ .
  43.  $X \times (Y - Z) = (X \times Y) - (X \times Z)$  for all sets  $X, Y$ , and  $Z$ .
  44.  $X - (Y \times Z) = (X - Y) \times (X - Z)$  for all sets  $X, Y$ , and  $Z$ .
  45.  $X \cap (Y \times Z) = (X \cap Y) \times (X \cap Z)$  for all sets  $X, Y$ , and  $Z$ .
  46. Prove the associative laws for sets [Theorem 1.1.22, part (a)].
  47. Prove the commutative laws for sets [Theorem 1.1.22, part (b)].
  48. Prove the distributive laws for sets [Theorem 1.1.22, part (c)].
  49. Prove the identity laws for sets [Theorem 1.1.22, part (d)].
  50. Prove the complement laws for sets [Theorem 1.1.22, part (e)].
  51. Prove the idempotent laws for sets [Theorem 1.1.22, part (f)].
  52. Prove the bound laws for sets [Theorem 1.1.22, part (g)].
  53. Prove the absorption laws for sets [Theorem 1.1.22, part (h)].
  54. Prove the involution law for sets [Theorem 1.1.22, part (i)].
  55. Prove the 0/1 laws for sets [Theorem 1.1.22, part (j)].
  56. Prove De Morgan's laws for sets [Theorem 1.1.22, part (k)].
- In Exercises 57–65,  $\Delta$  denotes the symmetric difference operator defined as  $A \Delta B = (A \cup B) - (A \cap B)$ , where  $A$  and  $B$  are sets.
57. Prove that  $A \Delta B = (A - B) \cup (B - A)$  for all sets  $A$  and  $B$ .
  58. Prove that  $(A \Delta B) \Delta A = B$  for all sets  $A$  and  $B$ .
  - \*59. Prove or disprove: If  $A, B$ , and  $C$  are sets satisfying  $A \Delta C = B \Delta C$ , then  $A = B$ .
  60. Prove or disprove:  $A \Delta (B \cup C) = (A \Delta B) \cup (A \Delta C)$  for all sets  $A, B$ , and  $C$ .
  61. Prove or disprove:  $A \Delta (B \cap C) = (A \Delta B) \cap (A \Delta C)$  for all sets  $A, B$ , and  $C$ .
  62. Prove or disprove:  $A \cup (B \Delta C) = (A \cup B) \Delta (A \cup C)$  for all sets  $A, B$ , and  $C$ .
  63. Prove or disprove:  $A \cap (B \Delta C) = (A \cap B) \Delta (A \cap C)$  for all sets  $A, B$ , and  $C$ .
  64. Is  $\Delta$  commutative? If so, prove it; otherwise, give a counterexample.
  - \*65. Is  $\Delta$  associative? If so, prove it; otherwise, give a counterexample.

## 2.2 More Methods of Proof

---

In this section, we discuss several more methods of proof: proof by contradiction, proof by contrapositive, proof by cases, proofs of equivalence, and existence proofs. We will find these proof techniques of use throughout this book.

### Proof by Contradiction

A **proof by contradiction** establishes  $p \rightarrow q$  by assuming that the hypothesis  $p$  is true and that the conclusion  $q$  is false and then, using  $p$  and  $\neg q$  as well as other axioms, definitions, previously derived theorems, and rules of inference, derives a **contradiction**. A contradiction is a proposition of the form  $r \wedge \neg r$  ( $r$  may be any proposition whatever). A proof by contradiction is sometimes called an **indirect proof** since to establish  $p \rightarrow q$  using proof by contradiction, we follow an indirect route: We derive  $r \wedge \neg r$  and then conclude that  $q$  is true.

The only difference between the assumptions in a direct proof and a proof by contradiction is the negated conclusion. In a direct proof the negated conclusion is *not* assumed, whereas in a proof by contradiction the negated conclusion *is* assumed.

Proof by contradiction may be justified by noting that the propositions  $p \rightarrow q$  and  $(p \wedge \neg q) \rightarrow (r \wedge \neg r)$  are equivalent. The equivalence is immediate from a truth table:

| $p$ | $q$ | $r$ | $p \rightarrow q$ | $p \wedge \neg q$ | $r \wedge \neg r$ | $(p \wedge \neg q) \rightarrow (r \wedge \neg r)$ |
|-----|-----|-----|-------------------|-------------------|-------------------|---------------------------------------------------|
| T   | T   | T   | T                 | F                 | F                 | T                                                 |
| T   | T   | F   | T                 | F                 | F                 | T                                                 |
| T   | F   | T   | F                 | T                 | F                 | F                                                 |
| T   | F   | F   | F                 | T                 | F                 | F                                                 |
| F   | T   | T   | T                 | F                 | F                 | T                                                 |
| F   | T   | F   | T                 | F                 | F                 | T                                                 |
| F   | F   | T   | T                 | F                 | F                 | T                                                 |
| F   | F   | F   | T                 | F                 | F                 | T                                                 |

#### Example 2.2.1

Give a proof by contradiction of the following statement:

For every  $n \in \mathbf{Z}$ , if  $n^2$  is even, then  $n$  is even.

**SOLUTION Discussion** First, let us consider giving a *direct* proof of this statement. We would assume the hypothesis, that is, that  $n^2$  is even. Then there exists an integer  $k_1$  such that  $n^2 = 2k_1$ . To prove that  $n$  is even, we must find an integer  $k_2$  such that  $n = 2k_2$ . It is not clear how to get from  $n^2 = 2k_1$  to  $n = 2k_2$ . (Taking the square root certainly does not work!) When a proof technique seems unpromising, try a different one.

In a proof by contradiction, we assume the hypothesis ( $n^2$  is even) *and* the negation of the conclusion ( $n$  is not even, i.e.,  $n$  is odd). Since  $n$  is odd, there exists an integer  $k$  such that  $n = 2k + 1$ . If we square both sides of this last equation, we obtain

$$n^2 = (2k + 1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1.$$

But this last equation tells us that  $n^2$  is odd. We have our contradiction:  $n^2$  is even (hypothesis) and  $n^2$  is odd. Formally, if  $r$  is the statement “ $n^2$  is even,” we have deduced  $r \wedge \neg r$ .

**Proof** We give a proof by contradiction. Thus we assume the hypothesis  $n^2$  is even and that the conclusion is false  $n$  is odd. Since  $n$  is odd, there exists an integer  $k$  such that  $n = 2k + 1$ . Now

$$n^2 = (2k+1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1.$$

Thus  $n^2$  is odd, which contradicts the hypothesis  $n^2$  is even. The proof by contradiction is complete. We have proved that for every  $n \in \mathbf{Z}$ , if  $n^2$  is even, then  $n$  is even. ◀ ◀

### Example 2.2.2

Give a proof by contradiction of the following statement:

For all real numbers  $x$  and  $y$ , if  $x + y \geq 2$ , then either  $x \geq 1$  or  $y \geq 1$ .

**SOLUTION Discussion** As in the previous example, a direct proof seems unpromising—assuming only that  $x + y \geq 2$  appears to be too little to get us started. We turn to a proof by contradiction.

**Proof** We begin by letting  $x$  and  $y$  be arbitrary real numbers. We then suppose that the conclusion is false, that is, that  $\neg(x \geq 1 \vee y \geq 1)$  is true. By De Morgan's laws of logic (see Example 1.3.11),

$$\neg(x \geq 1 \vee y \geq 1) \equiv \neg(x \geq 1) \wedge \neg(y \geq 1) \equiv (x < 1) \wedge (y < 1).$$

In words, we are assuming that  $x < 1$  and  $y < 1$ . Using a previous theorem, we may add these inequalities to obtain  $x + y < 1 + 1 = 2$ . At this point, we have derived a contradiction:  $x + y \geq 2$  and  $x + y < 2$ . Thus we conclude that for all real numbers  $x$  and  $y$ , if  $x + y \geq 2$ , then either  $x \geq 1$  or  $y \geq 1$ . ◀ ◀

### Example 2.2.3

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

**SOLUTION Discussion** Here a direct proof seems particularly bleak. It seems we have a blank slate with which to begin. However, if we use proof by contradiction, we may assume that  $\sqrt{2}$  is rational. In this case, we know that there exist integers  $p$  and  $q$  such that  $\sqrt{2} = p/q$ . Now we have an entry on our slate. We can manipulate this equation and hope to obtain a contradiction.

**Proof** We use proof by contradiction and assume that  $\sqrt{2}$  is rational. Then there exist integers  $p$  and  $q$  such that  $\sqrt{2} = p/q$ . We assume that the fraction  $p/q$  is in lowest terms so that  $p$  and  $q$  are not both even. Squaring  $\sqrt{2} = p/q$  gives  $2 = p^2/q^2$ , and multiplying by  $q^2$  gives  $2q^2 = p^2$ . It follows that  $p^2$  is even. Example 2.2.1 tells us that  $p$  is even. Therefore, there exists an integer  $k$  such that  $p = 2k$ . Substituting  $p = 2k$  into  $2q^2 = p^2$  gives  $2q^2 = (2k)^2 = 4k^2$ . Canceling 2 gives  $q^2 = 2k^2$ . Therefore  $q^2$  is even, and Example 2.2.1 tells us the  $q$  is even. Thus  $p$  and  $q$  are both even, which contradicts our assumption that  $p$  and  $q$  are not both even. Therefore,  $\sqrt{2}$  is irrational. ◀ ◀

### Proof by Contrapositive

Suppose that we give a proof by contradiction of  $p \rightarrow q$  in which, as in Examples 2.2.1 and 2.2.2, we deduce  $\neg p$ . In effect, we have proved  $\neg q \rightarrow \neg p$ . [Recall (see Theorem 1.3.18) that  $p \rightarrow q$  and  $\neg q \rightarrow \neg p$  are equivalent.] This special case of proof by contradiction is called **proof by contrapositive**.

### Example 2.2.4

Give a proof by contradiction that for any subset  $S$  of 26 cards from an ordinary 52-card deck (composed of four suits of 13 cards each), there is a suit such that  $S$  has at least 7 cards of that suit. (In the card game bridge, this result says that two partners, who each hold 13-card hands, will between them have a suit of at least 7 cards.)

**SOLUTION Discussion** The conclusion is: There is a suit such that  $S$  has at least 7 cards of that suit. If the conclusion is false, it must be that  $S$  has at most 6 cards of *every* suit. But, since there are four suits, we can account for at most  $6 \cdot 4 = 24$  cards in  $S$ . But the hypothesis is that  $S$  contains 26 cards. Contradiction! We have a proof.

We conclude the discussion by formally showing how De Morgan's laws of logic (see Example 1.3.1) are used to yield the negated conclusion. We take as our domain of discourse, the set of all subsets of cards from an ordinary 52-card deck. One piece of notation will be useful. If  $S_u$  is a suit, we let  $S(S_u)$  denote the number of cards of suit  $S_u$  in  $S$ . For example, the value of  $S(Club)$  is the number of clubs in  $S$ .

We are to prove that

$$\forall S, \text{ if } |S| = 26, \text{ then } \exists \text{ a suit } S_u \text{ such that } S(S_u) \geq 7.$$

By De Morgan's laws of logic,

$$\neg(\exists \text{ a suit } S_u \text{ such that } S(S_u) \geq 7) \equiv \forall \text{ suits } S_u, S(S_u) < 7;$$

that is, the negation of the conclusion can be written:  $S$  has less than 7 (i.e., at most 6) cards of every suit.

In Example 2.2.8, we will discuss an alternative approach to finding a proof of this result.

**Proof** We are given a subset  $S$  of 26 cards from an ordinary 52-card deck. Suppose by way of contradiction that the conclusion is false; that is, suppose that  $S$  has at most 6 cards of every suit. Since there are four suits,  $S$  has at most  $6 \cdot 4 = 24$  cards. This contradicts the fact that  $S$  has 26 cards. Therefore, there is a suit such that  $S$  has at least 7 cards of that suit. ◀ ◀

### Example 2.2.5

Give a proof by contrapositive to prove that

$$\text{for all } x \in \mathbb{R}, \text{ if } x^2 \text{ is irrational, then } x \text{ is irrational.}$$

**SOLUTION Discussion** For much the same reasons that a direct proof seemed unpromising in Examples 2.2.1 and 2.2.2, a direct proof in which we assume only that  $x^2$  is irrational seems to be too little to get us started. A proof by contradiction can be devised (see Exercise 1), but here a proof by contrapositive is requested.

**Proof** We begin by letting  $x$  be an arbitrary real number. We prove the contrapositive of the given statement, which is

$$\text{if } x \text{ is not irrational, then } x^2 \text{ is not irrational}$$

or, equivalently,

$$\text{if } x \text{ is rational, then } x^2 \text{ is rational.}$$

So suppose that  $x$  is rational. Then  $x = p/q$  for some integers  $p$  and  $q$ . Now  $x^2 = p^2/q^2$ . Since  $x^2$  is the quotient of integers,  $x^2$  is rational. The proof is complete. ◀ ◀

### Proof by Cases

**Proof by cases** is used when the original hypothesis naturally divides itself into various cases. For example, the hypothesis “ $x$  is a real number” can be divided into cases: (a)  $x$

is a nonnegative real number and (b)  $x$  is a negative real number. Suppose that the task is to prove  $p \rightarrow q$  and that  $p$  is equivalent to  $p_1 \vee p_2 \vee \cdots \vee p_n$  ( $p_1, \dots, p_n$  are the cases). Instead of proving

$$(p_1 \vee p_2 \vee \cdots \vee p_n) \rightarrow q, \quad (2.2.1)$$

we prove

$$(p_1 \rightarrow q) \wedge (p_2 \rightarrow q) \wedge \cdots \wedge (p_n \rightarrow q). \quad (2.2.2)$$

As we will show, proof by cases is justified because the two statements are equivalent.

First suppose that  $q$  is true. Then all the implications in (2.2.1) and (2.2.2) are true (regardless of the truth value of the hypotheses). Thus (2.2.1) and (2.2.2) are true.

Now suppose that  $q$  is false. If all the  $p_i$  are false, then all the implications in (2.2.1) and (2.2.2) are true, so (2.2.1) and (2.2.2) are true. If for some  $j$ ,  $p_j$  is true, then  $p_1 \vee \cdots \vee p_n$  is true, so (2.2.1) is false. Since  $p_j \rightarrow q$  is false, (2.2.2) is false. Thus (2.2.1) and (2.2.2) are false. Therefore, (2.2.1) and (2.2.2) are equivalent.

Sometimes the number of cases to prove is finite and not too large, so we can check them all one by one. We call this type of proof **exhaustive proof**.

### Example 2.2.6

Prove that  $2m^2 + 3n^2 = 40$  has no solution in positive integers, that is, that  $2m^2 + 3n^2 = 40$  is false for all positive integers  $m$  and  $n$ .

**SOLUTION Discussion** We certainly *cannot* check  $2m^2 + 3n^2$  for *all* positive integers  $m$  and  $n$ , but we can rule out most positive integers because, if  $2m^2 + 3n^2 = 40$ , the sizes of  $m$  and  $n$  are restricted. In particular, we must have  $2m^2 \leq 40$  and  $3n^2 \leq 40$ . (If, for example,  $2m^2 > 40$ , when we add  $3n^2$  to  $2m^2$ , the sum  $2m^2 + 3n^2$  will exceed 40.) If  $2m^2 \leq 40$ , then  $m^2 \leq 20$  and  $m$  can be at most 4. Similarly, if  $3n^2 \leq 40$ , then  $n^2 \leq 40/3$  and  $n$  can be at most 3. Thus it suffices to check the cases  $m = 1, 2, 3, 4$  and  $n = 1, 2, 3$ .

**Proof** If  $2m^2 + 3n^2 = 40$ , we must have  $2m^2 \leq 40$ . Thus  $m^2 \leq 20$  and  $m \leq 4$ . Similarly, we must have  $3n^2 \leq 40$ . Thus  $n^2 \leq 40/3$  and  $n \leq 3$ . Therefore it suffices to check the cases  $m = 1, 2, 3, 4$  and  $n = 1, 2, 3$ .

The entries in the table give the value of  $2m^2 + 3n^2$  for the indicated values of  $m$  and  $n$ .

|     |   | $m$ |    |    |    |
|-----|---|-----|----|----|----|
|     |   | 1   | 2  | 3  | 4  |
| $n$ | 1 | 5   | 11 | 21 | 35 |
|     | 2 | 14  | 20 | 30 | 44 |
|     | 3 | 29  | 35 | 45 | 59 |

Since  $2m^2 + 3n^2 \neq 40$  for  $m = 1, 2, 3, 4$  and  $n = 1, 2, 3$ , and  $2m^2 + 3n^2 > 40$  for  $m > 4$  or  $n > 3$ , we conclude that  $2m^2 + 3n^2 = 40$  has no solution in positive integers. ◀ ◀

Example 2.2.7 illustrates an important point: An *or* statement often leads itself to a proof by cases.

### Example 2.2.7

We prove that for every real number  $x$ ,  $x \leq |x|$ .

**SOLUTION Discussion** Since  $x$  is a real number, either  $x \geq 0$  or  $x < 0$ . We use this *or* statement to divide the proof into cases. We divide the proof into cases because

the definition of absolute value is itself divided into cases  $x \geq 0$  and  $x < 0$  (see Example 2.1.2). Case 1 is  $x \geq 0$  and case 2 is  $x < 0$ .

**Proof** If  $x \geq 0$ , by definition  $|x| = x$ . Thus  $|x| \geq x$ . If  $x < 0$ , by definition  $|x| = -x$ . Since  $|x| = -x > 0$  and  $0 > x$ ,  $|x| \geq x$ . In either case,  $|x| \geq x$ ; so the proof is complete. ◀ ◀

Example 2.2.8 offers an alternative approach to developing a proof of the result in Example 2.2.4. There are often many approaches and proofs of a single result.

### Example 2.2.8

We revisit the problem (see Example 2.2.4) of proving that for any subset  $S$  of 26 cards from an ordinary 52-card deck, there is a suit such that  $S$  has at least 7 cards of that suit.

**Discussion** Consider trying a direct proof. How about the club suit? If there are 7 or more clubs, we have the desired conclusion. What if there are 6 or fewer clubs? Try another suit! Let us try diamonds next. If there are 7 or more diamonds, we have the desired conclusion. What if there are 6 or fewer diamonds? Try another suit, and so on. If each of the four suits consists of 6 or fewer cards, we cannot deduce the conclusion; however, if each suit has 6 or fewer cards, we can account for only 24 cards. But there are 26 cards; this case cannot occur.

The preceding discussion shows that we can divide the proof into two cases. Case 1 is that some suit consists of 7 or more cards. (Proof complete!) Case 2 is that every suit consists of 6 or fewer cards. (We have shown that this case cannot occur.) Since only case 1 can occur, the proof is complete.

We started by thinking along the lines of a direct proof, which led us to a proof by cases. The proof by cases is essentially identical to the proof by contradiction in Example 2.2.4.

**Proof** Let  $S$  be a subset of 26 cards from an ordinary 52-card deck. We consider two cases. Case 1 is that there is a suit such that  $S$  has at least 7 or more cards of that suit. Case 2 is that  $S$  has 6 or fewer cards of every suit.

If case 1 holds, the proof is complete. Suppose that case 2 holds. Since  $S$  has 6 or fewer cards of each of the four suits,  $S$  has at most 24 cards. But we are given that  $S$  has 26 cards. Therefore, case 2 cannot hold. Since only case 1 holds, the proof is complete. ◀ ◀

### Proofs of Equivalence

Some theorems are of the form  $p$  if and only if  $q$ . Such theorems are proved by using the equivalence (see Example 1.3.15)

$$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p);$$

that is, to prove “ $p$  if and only if  $q$ ,” prove “if  $p$  then  $q$ ” and “if  $q$  then  $p$ .”

### Example 2.2.9

Prove that for every integer  $n$ ,  $n$  is odd if and only if  $n - 1$  is even.

**SOLUTION Discussion** We let  $n$  be an arbitrary integer. We must prove that

$$\text{if } n \text{ is odd then } n - 1 \text{ is even} \quad (2.2.3)$$

and

$$\text{if } n - 1 \text{ is even then } n \text{ is odd.} \quad (2.2.4)$$

**Proof** We first prove (2.2.3). If  $n$  is odd, then  $n = 2k + 1$  for some integer  $k$ . Now  $n - 1 = (2k + 1) - 1 = 2k$ . Therefore,  $n - 1$  is even.

Next we prove (2.2.4). If  $n - 1$  is even, then  $n - 1 = 2k$  for some integer  $k$ . Now  $n = 2k + 1$ . Therefore,  $n$  is odd. The proof is complete. ◀ ◀

Some proofs of  $p \leftrightarrow q$  combine the proofs of  $p \rightarrow q$  and  $q \rightarrow p$ . For example, the proof in Example 2.2.9 could be written as follows:

$n$  is odd if and only if  $n = 2k + 1$  for some integer  $k$  if and only if  $n - 1 = 2k$  for some integer  $k$  if and only if  $n - 1$  is even.

For such a proof to be correct, it must be the case that each if-and-only-if statement is true. If such a proof of  $p \leftrightarrow q$  is read in one direction, we obtain the proof of  $p \rightarrow q$  and, if it is read in the other direction, we obtain a proof of  $q \rightarrow p$ . Reading the preceding proof in the if-then direction,

if  $n$  is odd, then  $n = 2k + 1$  for some integer  $k$ ; if  $n = 2k + 1$  for some integer  $k$ , then  $n - 1 = 2k$  for some integer  $k$ ; if  $n - 1 = 2k$  for some integer  $k$ , then  $n - 1$  is even,

proves that if  $n$  is odd, then  $n - 1$  is even. Reversing the order,

if  $n - 1$  is even, then  $n - 1 = 2k$  for some integer  $k$ ; if  $n - 1 = 2k$  for some integer  $k$ , then  $n = 2k + 1$  for some integer  $k$ ; if  $n = 2k + 1$  for some integer  $k$ , then  $n$  is odd,

proves that if  $n - 1$  is even, then  $n$  is odd.

### Example 2.2.10

Prove that for all real numbers  $x$  and all positive real numbers  $d$ ,

$$|x| < d \text{ if and only if } -d < x < d.$$

**SOLUTION Discussion** We let  $x$  be an arbitrary real number and  $d$  be an arbitrary positive real number. We must show

$$\text{if } |x| < d \text{ then } -d < x < d \quad (2.2.5)$$

and

$$\text{if } -d < x < d \text{ then } |x| < d. \quad (2.2.6)$$

Since  $|x|$  is defined by cases, we expect to use proof by cases.

**Proof** To show (2.2.5), we use proof by cases. We assume that  $|x| < d$ . If  $x \geq 0$ , then  $-d < 0 \leq x = |x| < d$ . If  $x < 0$ , then  $-d < 0 < -x = |x| < d$ ; that is,  $-d < -x < d$ . Multiplying by  $-1$ , we obtain  $d > x > -d$ . In either case, we have proved that  $-d < x < d$ .

To show (2.2.6), we also use proof by cases. We assume that  $-d < x < d$ . If  $x \geq 0$ , then  $|x| = x < d$ . If  $x < 0$ , then  $|x| = -x$ . Since  $-d < x$ , we may multiply by  $-1$  to obtain  $d > -x$ . Combining  $|x| = -x$  and  $d > -x$  gives  $|x| = -x < d$ . In either case, we have proved that  $|x| < d$ . The proof is complete. ◀ ◀

In proving  $p \leftrightarrow q$ , we are proving that  $p$  and  $q$  are logically equivalent, that is,  $p$  and  $q$  are either both true or both false. Some theorems state that three or more statements are logically equivalent and, thus, have the form

The following are equivalent:

- (a) —
- (b) —
- (c) —
- ⋮

Such a theorem asserts that (a), (b), (c), and so on are either all true or all false.

To prove that  $p_1, p_2, \dots, p_n$  are equivalent, the usual method is to prove

$$(p_1 \rightarrow p_2) \wedge (p_2 \rightarrow p_3) \wedge \cdots \wedge (p_{n-1} \rightarrow p_n) \wedge (p_n \rightarrow p_1). \quad (2.2.7)$$

We show that proving (2.2.7) shows that  $p_1, p_2, \dots, p_n$  are equivalent.

Suppose that we prove (2.2.7). We consider two cases:  $p_1$  is true,  $p_1$  is false. First, suppose that  $p_1$  is true. Because  $p_1$  and  $p_1 \rightarrow p_2$  are true,  $p_2$  is true; because  $p_2$  and  $p_2 \rightarrow p_3$  are true,  $p_3$  is true; and so on. In this case,  $p_1, p_2, \dots, p_n$  have the same truth value: each is true.

Now suppose that  $p_1$  is false. Because  $p_1$  is false and  $p_n \rightarrow p_1$  is true,  $p_n$  is false; because  $p_n$  is false and  $p_{n-1} \rightarrow p_n$  is true,  $p_{n-1}$  is false; and so on. In this case,  $p_1, p_2, \dots, p_n$  have the same truth value; each is false. Therefore, proving (2.2.7) shows that  $p_1, p_2, \dots, p_n$  are equivalent.

Not just any arrangement of implications along the lines of (2.3.7) will make  $p_1, \dots, p_n$  equivalent. For example, consider  $p_1: 2 = 3$ ,  $p_2: 4 = 6$ , and  $p_3: 8 = 8$ . For the arrangement

$$p_1 \rightarrow p_2, \quad p_2 \rightarrow p_3, \quad p_1 \rightarrow p_3,$$

$p_1 \rightarrow p_2, p_2 \rightarrow p_3$ , and  $p_1 \rightarrow p_3$  are all true, but  $p_1, p_2, p_3$  are *not* equivalent.

### Example 2.2.11

Let  $A$ ,  $B$ , and  $C$  be sets. Prove that the following are equivalent:

- (a)  $A \subseteq B$
- (b)  $A \cap B = A$
- (c)  $A \cup B = B$ .

**SOLUTION** **Discussion** According to the discussion preceding this example, we must prove

$$[(a) \rightarrow (b)] \wedge [(b) \rightarrow (c)] \wedge [(c) \rightarrow (a)].$$

**Proof** We prove  $(a) \rightarrow (b)$ ,  $(b) \rightarrow (c)$ , and  $(c) \rightarrow (a)$ .

$[(a) \rightarrow (b)]$  We assume that  $A \subseteq B$ , and prove that  $A \cap B = A$ . Suppose that  $x \in A \cap B$ . We must show that  $x \in A$ . But if  $x \in A \cap B$ ,  $x \in A$  by the definition of intersection.

Now suppose that  $x \in A$ . We must show that  $x \in A \cap B$ . Since  $A \subseteq B$ ,  $x \in B$ . Therefore  $x \in A \cap B$ . We have proved that  $A \cap B = A$ .

$[(b) \rightarrow (c)]$  We assume that  $A \cap B = A$ , and prove that  $A \cup B = B$ . Suppose that  $x \in A \cup B$ . We must show that  $x \in B$ . By assumption, either  $x \in A$  or  $x \in B$ . If  $x \in B$ , we have the desired conclusion. If  $x \in A$ , since  $A \cap B = A$ , again  $x \in B$ .

Now suppose that  $x \in B$ . We must show that  $x \in A \cup B$ . But if  $x \in B$ ,  $x \in A \cup B$  by the definition of union. We have proved that  $A \cup B = B$ .

$[(c) \rightarrow (a)]$  We assume that  $A \cup B = B$ , and prove that  $A \subseteq B$ . Suppose that  $x \in A$ . We must show that  $x \in B$ . Since  $x \in A$ ,  $x \in A \cup B$  by the definition of union. Since  $A \cup B = B$ ,  $x \in B$ . We have proved that  $A \subseteq B$ . The proof is complete. 

**Existence Proofs**

A proof of

$$\exists x P(x) \quad (2.2.8)$$

is called an **existence proof**. In Section 1.5, we showed that one way to prove (2.2.8) is to exhibit one member  $a$  in the domain of discourse that makes  $P(a)$  true.

**Example 2.2.12**

Let  $a$  and  $b$  be real numbers with  $a < b$ . Prove that there exists a real number  $x$  satisfying  $a < x < b$ .

**SOLUTION Discussion** We will prove the statement by exhibiting a real number  $x$  between  $a$  and  $b$ . The point midway between  $a$  and  $b$  suffices.

**Proof** It suffices to find one real number  $x$  satisfying  $a < x < b$ . The real number

$$x = \frac{a+b}{2},$$

halfway between  $a$  and  $b$ , surely satisfies  $a < x < b$ . ◀ ◀

**Example 2.2.13**

Prove that there exists a prime  $p$  such that  $2^p - 1$  is composite (i.e., *not* prime).

**SOLUTION Discussion** By trial and error, we find that  $2^p - 1$  is prime for  $p = 2, 3, 5, 7$  but not  $p = 11$  since  $2^{11} - 1 = 2048 - 1 = 2047 = 23 \cdot 89$ . Thus  $p = 11$  makes the given statement true.

**Proof** For the prime  $p = 11$ ,  $2^p - 1$  is composite:

$$2^{11} - 1 = 2048 - 1 = 2047 = 23 \cdot 89. \quad \blacktriangleleft \quad \blacktriangleright$$

**Go Online**

The Great Internet Mersenne Prime Search is at  
[goo.gl/gHgyey](http://goo.gl/gHgyey)

A prime number of the form  $2^p - 1$ , where  $p$  is prime, is called a *Mersenne prime* [named for Marin Mersenne (1588–1648)]. It is not known whether the number of Mersenne primes is finite or infinite. The largest primes known are Mersenne primes. In January 2016, the 49th known Mersenne prime was found,  $2^{74,207,281} - 1$ , a number having 22,338,618 decimal digits. This number was found by the Great Internet Mersenne Prime Search (GIMPS). GIMPS is a computer program distributed over many personal computers maintained by volunteers. You can participate. Just check the web link. You may find the next Mersenne prime!

An existence proof of (2.2.8) that exhibits an element  $a$  of the domain of discourse that makes  $P(a)$  true is called a **constructive proof**. The proofs in Examples 2.2.12 and 2.2.13 are constructive proofs. A proof of (2.2.8) that does not exhibit an element  $a$  of the domain of discourse that makes  $P(a)$  true, but rather proves (2.2.8) some other way (e.g., using proof by contradiction), is called a **nonconstructive proof**.

**Example 2.2.14**

Let

$$A = \frac{s_1 + s_2 + \cdots + s_n}{n}$$

be the average of the real numbers  $s_1, \dots, s_n$ . Prove that there exists  $i$  such that  $s_i \geq A$ .

**SOLUTION Discussion** It seems hopeless to choose an  $i$  and prove that  $s_i \geq A$ ; instead, we use proof by contradiction.

**Proof** We use proof by contradiction and assume the negation of the conclusion  $\neg\exists i(s_i \geq A)$ . By the generalized De Morgan's laws for logic (Theorem 1.5.14), the negation of the conclusion is equivalent to  $\forall i\neg(s_i \geq A)$  or  $\forall i(s_i < A)$ . Thus we assume  $s_1 < A, s_2 < A, \dots, s_n < A$ . Adding these inequalities yields

$$s_1 + s_2 + \cdots + s_n < nA.$$

Dividing by  $n$  gives

$$\frac{s_1 + s_2 + \cdots + s_n}{n} < A,$$

which contradicts the hypothesis

$$A = \frac{s_1 + s_2 + \cdots + s_n}{n}.$$

Therefore, there exists  $i$  such that  $s_i \geq A$ . ◀ ◀

The proof in Example 2.2.14 is nonconstructive; it does not exhibit an  $i$  for which  $s_i \geq A$ . It does however *prove* indirectly using proof by contradiction that there is such an  $i$ . We could find such an  $i$ : We could check whether  $s_1 \geq A$  and if true, stop. Otherwise, we could check whether  $s_2 \geq A$  and if true, stop. We could continue in this manner until we find an  $i$  for which  $s_i \geq A$ . Example 2.2.14 guarantees that there is such an  $i$ .

## 2.2 Problem-Solving Tips

It is worth reviewing the Problem-Solving Tips of Section 2.1. Tips specific to the present section follow.

- If you are trying to construct a direct proof of a statement of the form  $p \rightarrow q$  and you seem to be getting stuck, try a proof by contradiction. You then have more to work with: Besides assuming  $p$ , you get to assume  $\neg q$ .
- When writing up a proof by contradiction, alert the reader by stating, “We give a proof by contradiction, thus we assume  $\dots$ ,” where  $\dots$  is the negation of the conclusion. Another common introduction is: Assume by way of contradiction that  $\dots$ .
- Proof by cases is useful if the hypotheses naturally break down into parts. For example, if the statement to prove involves the absolute value of  $x$ , you may want to consider the cases  $x \geq 0$  and  $x < 0$  because  $|x|$  is itself defined by the cases  $x \geq 0$  and  $x < 0$ . If the number of cases to prove is finite and not too large, the cases can be directly checked one by one.

In writing up a proof by cases, it is sometimes helpful to the reader to indicate the cases, for example,

[Case I:  $x \geq 0$ .] Proof of this case goes here.

[Case II:  $x < 0$ .] Proof of this case goes here.

- To prove  $p$  if and only if  $q$ , you must prove two statements: (1) if  $p$  then  $q$  and (2) if  $q$  then  $p$ . It helps the reader if you state clearly what you are proving. You can write up the proof of (1) by beginning a new paragraph with a sentence that indicates that you are about to prove “if  $p$  then  $q$ .” You would then follow with a proof of (2) by beginning a new paragraph with a sentence that indicates that you are about to prove “if  $q$  then  $p$ .” Another common technique is to write

[ $p \rightarrow q$ .] Proof of  $p \rightarrow q$  goes here.

[ $q \rightarrow p$ .] Proof of  $q \rightarrow p$  goes here.

- To prove that several statements, say  $p_1, \dots, p_n$ , are equivalent, prove  $p_1 \rightarrow p_2, p_2 \rightarrow p_3, \dots, p_{n-1} \rightarrow p_n, p_n \rightarrow p_1$ . The statements can be ordered in any way and the proofs may be easier to construct for one ordering than another. For example, you could swap  $p_2$  and  $p_3$  and prove  $p_1 \rightarrow p_3, p_3 \rightarrow p_2, p_2 \rightarrow p_4, p_4 \rightarrow p_5, \dots, p_{n-1} \rightarrow p_n, p_n \rightarrow p_1$ . You should indicate clearly what you are about to prove. One common form is

[ $p_1 \rightarrow p_2$ .] Proof of  $p_1 \rightarrow p_2$  goes here.

[ $p_2 \rightarrow p_3$ .] Proof of  $p_2 \rightarrow p_3$  goes here.

And so forth.

- If the statement is existentially quantified (i.e., there exists  $x \dots$ ), the proof, called an existence proof, consists of showing that there exists at least one  $x$  in the domain of discourse that makes the statement true. One type of existence proof exhibits a value of  $x$  that makes the statement true (and proves that the statement is indeed true for the specific  $x$ ). Another type of existence proof indirectly proves (e.g., using proof by contradiction) that a value of  $x$  exists that makes the statement true without specifying any particular value of  $x$  for which the statement is true.

## 2.2 Review Exercises

- What is proof by contradiction?
- Give an example of a proof by contradiction.
- What is an indirect proof?
- What is proof by contrapositive?
- Give an example of a proof by contrapositive.
- What is proof by cases?
- Give an example of a proof by cases.
- What is a proof of equivalence?
- Give an example of a proof of equivalence.
- How can we show that three statements, say (a), (b), and (c), are equivalent?
- What is an existence proof?
- What is a constructive existence proof?
- Give an example of a constructive existence proof.
- What is a nonconstructive existence proof?
- Give an example of a nonconstructive existence proof.

## 2.2 Exercises

- Use proof by contradiction to prove that for all  $x \in \mathbf{R}$ , if  $x^2$  is irrational, then  $x$  is irrational.
- Is the converse of Exercise 1 true or false? Prove your answer.
- Prove that for all  $x \in \mathbf{R}$ , if  $x^3$  is irrational, then  $x$  is irrational.
- Prove that for every  $n \in \mathbf{Z}$ , if  $n^2$  is odd, then  $n$  is odd.
- Prove that for all real numbers  $x, y$ , and  $z$ , if  $x + y + z \geq 3$ , then either  $x \geq 1$  or  $y \geq 1$  or  $z \geq 1$ .
- Prove that for all real numbers  $x$  and  $y$ , if  $xy \leq 2$ , then either  $x \leq \sqrt{2}$  or  $y \leq \sqrt{2}$ .
- Prove that  $\sqrt[3]{2}$  is irrational.
- Prove that for all  $x, y \in \mathbf{R}$ , if  $x$  is rational and  $y$  is irrational, then  $x + y$  is irrational.
- Prove or disprove: For all  $x, y \in \mathbf{R}$ , if  $x$  is rational and  $y$  is irrational, then  $xy$  is irrational.
- Prove that if  $a$  and  $b$  are real numbers with  $a < b$ , there exists a rational number  $x$  satisfying  $a < x < b$ .
- Fill in the details of the following proof that there exist irrational numbers  $a$  and  $b$  such that  $a^b$  is rational.

**Proof** Let  $x = y = \sqrt{2}$ . If  $x^y$  is rational, the proof is complete. (Explain.) Otherwise, suppose that  $x^y$  is irrational. (Why?) Let  $a = x^y$  and  $b = \sqrt{2}$ . Consider  $a^b$ . (How does this complete the proof?)

Is this proof constructive or nonconstructive?

- Prove or disprove: There exist rational numbers  $a$  and  $b$  such that  $a^b$  is rational. What kind of proof did you give?
- Prove or disprove: There exist rational numbers  $a$  and  $b$  such that  $a^b$  is irrational. What kind of proof did you give?
- Let  $x$  and  $y$  be real numbers. Prove that if  $x \leq y + \varepsilon$  for every positive real number  $\varepsilon$ , then  $x \leq y$ .
- In American football, a safety counts two points, a field goal three points, a touchdown six points, a successful kick imme-

diatey after a touchdown one point, and a successful run or pass immediately after a touchdown two points. What are the possible points a team can score? Prove your answer.

17. Suppose that a real number  $a$  has the property that  $a^n$  is irrational for every positive integer  $n$ . Prove that  $a^r$  is irrational for every positive rational number  $r$ . (There are real numbers such as  $a$ . The real number  $e = 2.718\dots$ , the base of the natural logarithm, has the property that  $e^r$  is irrational for every rational number  $r$ . An elementary proof, using only calculus, may be found in [Aigner].)
  18. Abby, Bosco, Cary, Dale, and Edie are considering going to hear the Flying Squirrels. If Abby goes, Cary will also go. If Bosco goes, Cary and Dale will also go. If Cary goes, Edie will also go. If Dale goes, Abby will also go. If Edie goes, Abby will also go. Exactly four of the five went to the concert. Who went? Prove your result.
  19. Prove or disprove:  $(X - Y) \cap (Y - X) = \emptyset$  for all sets  $X$  and  $Y$ .
  20. Prove or disprove:  $X \times \emptyset = \emptyset$  for every set  $X$ .
  21. Show, by giving a proof by contradiction, that if 100 balls are placed in nine boxes, some box contains 12 or more balls.
  22. Show, by giving a proof by contradiction, that if 40 coins are distributed among nine bags so that each bag contains at least 1 coin, at least two bags contain the same number of coins.
  23. Let  $S$  be a subset of 26 cards from an ordinary 52-card deck. Suppose that there is a suit in which  $S$  has exactly 7 cards. Prove that there is another suit in which  $S$  has at least 7 cards.
  24. Let  $S_1$  be a subset of 26 cards from an ordinary 52-card deck, and let  $S_2$  be the remaining 26 cards. Suppose that there is a suit in which  $S_1$  has at least 9 cards. Prove that there is a suit in which  $S_2$  has at least 8 cards. (In the card game bridge, two pairs compete against each other and initially hold 13 cards each. This result says a pair, who between them have a suit of at least 9 cards, will have opponents who between them have a suit of at least 8 cards.)
  25. Let  $s_1, \dots, s_n$  be a sequence<sup>†</sup> satisfying
    - $s_1$  is a positive integer and  $s_n$  is a negative integer,
    - for all  $i$ ,  $1 \leq i < n$ ,  $s_{i+1} = s_i + 1$  or  $s_{i+1} = s_i - 1$ .
 Prove that there exists  $i$ ,  $1 < i < n$ , such that  $s_i = 0$ .
  - Calculus students will recognize this exercise as a discrete version of the calculus theorem: If  $f$  is a continuous function on  $[a, b]$  and  $f(a) > 0$  and  $f(b) < 0$ , then  $f(c) = 0$  for some  $c$  in  $(a, b)$ . There are similar proofs of the two statements.
  - Disprove the statement: For every positive integer  $n$ ,  $n^2 \leq 2^n$ .
- In Exercises 27–31,
- $$A = \frac{s_1 + s_2 + \cdots + s_n}{n}$$
- is the average of the real numbers  $s_1, \dots, s_n$ .
27. Prove that there exists  $i$  such that  $s_i \leq A$ .
  28. Prove or disprove: There exists  $i$  such that  $s_i > A$ . What proof technique did you use?
  29. Suppose that there exists  $i$  such that  $s_i < A$ . Prove or disprove: There exists  $j$  such that  $s_j > A$ . What proof technique did you use?
  30. Suppose that there exist  $i$  and  $j$  such that  $s_i \neq s_j$ . Prove that there exists  $k$  such that  $s_k < A$ .
  31. Suppose that there exist  $i$  and  $j$  such that  $s_i \neq s_j$ . Prove that there exists  $k$  such that  $s_k > A$ .
  32. Prove that  $2m + 5n^2 = 20$  has no solution in positive integers.
  33. Prove that  $m^3 + 2n^2 = 36$  has no solution in positive integers.
  34. Prove that  $2m^2 + 4n^2 - 1 = 2(m+n)$  has no solution in positive integers.
  35. Prove that the product of two consecutive integers is even.
  36. Prove that for every  $n \in \mathbf{Z}$ ,  $n^3 + n$  is even.
  37. Use proof by cases to prove that  $|xy| = |x||y|$  for all real numbers  $x$  and  $y$ .
  38. Use proof by cases to prove that  $|x+y| \leq |x| + |y|$  for all real numbers  $x$  and  $y$ .
  39. Define the *sign of the real number*  $x$ ,  $\operatorname{sgn}(x)$ , as
- $$\operatorname{sgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0. \end{cases}$$
- Use proof by cases to prove that  $|x| = \operatorname{sgn}(x)x$  for every real number  $x$ .
40. Use proof by cases to prove that  $\operatorname{sgn}(xy) = \operatorname{sgn}(x)\operatorname{sgn}(y)$  for all real numbers  $x$  and  $y$  ( $\operatorname{sgn}$  is defined in Exercise 39).
  41. Use Exercises 39 and 40 to give another proof that  $|xy| = |x||y|$  for all real numbers  $x$  and  $y$ .
  42. Use proof by cases to prove that  $\max\{x, y\} + \min\{x, y\} = x + y$  for all real numbers  $x$  and  $y$ .
  43. Use proof by cases to prove that
- $$\max\{x, y\} = \frac{x + y + |x - y|}{2}$$
- for all real numbers  $x$  and  $y$ .
44. Use proof by cases to prove that
- $$\min\{x, y\} = \frac{x + y - |x - y|}{2}$$
- for all real numbers  $x$  and  $y$ .
45. Use Exercises 43 and 44 to prove that  $\max\{x, y\} + \min\{x, y\} = x + y$  for all real numbers  $x$  and  $y$ .
  46. Prove that for all  $n \in \mathbf{Z}$ ,  $n$  is even if and only if  $n + 2$  is even.
  47. Prove that for all  $n \in \mathbf{Z}$ ,  $n$  is odd if and only if  $n + 2$  is odd.
  48. Prove that for all sets  $A$  and  $B$ ,  $A \subseteq B$  if and only if  $\overline{B} \subseteq \overline{A}$ .

<sup>†</sup>Informally, a *sequence* is a list of elements in which order is taken into account, so that  $s_1$  is the first element,  $s_2$  is the second element, and so on. We present the formal definition in Section 3.2.

49. Prove that for all sets  $A$ ,  $B$ , and  $C$ ,  $A \subseteq C$  and  $B \subseteq C$  if and only if  $A \cup B \subseteq C$ .
50. Prove that for all sets  $A$ ,  $B$ , and  $C$ ,  $C \subseteq A$  and  $C \subseteq B$  if and only if  $C \subseteq A \cap B$ .
- \*51. The ordered pair  $(a, b)$  can be *defined* in terms of sets as  

$$(a, b) = \{\{a\}, \{a, b\}\}.$$
- Taking the preceding equation as the *definition* of ordered pair, prove that  $(a, b) = (c, d)$  if and only if  $a = c$  and  $b = d$ .
52. Prove that the following are equivalent for the integer  $n$ :
- (a)  $n$  is odd.    (b) There exists  $k \in \mathbf{Z}$  such that  $n = 2k - 1$ .  
(c)  $n^2 + 1$  is even.
53. Prove that the following are equivalent for sets  $A$ ,  $B$ , and  $C$ :
- (a)  $A \cap B = \emptyset$     (b)  $B \subseteq \overline{A}$     (c)  $A \Delta B = A \cup B$ ,
- where  $\Delta$  is the symmetric difference operator (see Exercise 101, Section 1.1).
54. Prove that the following are equivalent for sets  $A$ ,  $B$ , and  $C$ :
- (a)  $A \cup B = U$     (b)  $\overline{A} \cap \overline{B} = \emptyset$     (c)  $\overline{A} \subseteq B$ ,
- where  $U$  is a universal set.

## Problem-Solving Corner Proving Some Properties of Real Numbers

### Problem

First some definitions:

- (a) Let  $X$  be a nonempty set of real numbers. An *upper bound for  $X$*  is a real number  $a$  having the property that  $x \leq a$  for every  $x \in X$ .
- (b) Let  $a$  be an upper bound for a set  $X$  of real numbers. If every upper bound  $b$  for  $X$  satisfies  $b \geq a$ , we call  $a$  a *least upper bound for  $X$* .

A fundamental property of the real numbers is that every nonempty subset of real numbers bounded above has a least upper bound.

Answer the following where  $\mathbf{R}$  serves as a universal set:

- Give an example of a set  $X$  and three distinct upper bounds for  $X$ , one of which is a least upper bound for  $X$ .
- Prove that if  $a$  and  $b$  are least upper bounds for a set  $X$ , then  $a = b$ . We say that the least upper bound for a set  $X$  is *unique*. If  $a$  is the least upper bound of a set  $X$ , we sometimes write  $a = \text{lub } X$ .
- Let  $X$  be a set with least upper bound  $a$ . Prove that if  $\varepsilon > 0$ , then there exists  $x \in X$  satisfying  $a - \varepsilon < x \leq a$ .
- Let  $X$  be a set with least upper bound  $a$ , and suppose that  $t > 0$ . Prove that  $ta$  is the least upper bound of the set  $\{tx \mid x \in X\}$ .

### Attacking the Problem

To better understand the definitions, let's construct examples, write out the definitions in words, look at negations of the definitions, and draw pictures.

We'll start with definition (a) and construct a simple example—taking  $X$  to be a small finite set, say

$X = \{1, 2, 3, 4\}$ . Now an upper bound  $a$  for  $X$  satisfies  $x \leq a$  for every  $x$  in  $X$ —here, we must have

$$1 \leq a, \quad 2 \leq a, \quad 3 \leq a, \quad 4 \leq a.$$

Examples of upper bounds for  $X$  are 4, 6.9,  $3\pi$ , 9072.

In words, definition (a) says that  $a$  is an upper bound for a set  $X$  if every element in  $X$  is less than or equal to  $a$ . We see that upper bounds  $e$ ,  $f$ , and  $g$  for a set  $X$  (shown in color) look like



What would it mean that  $a$  is *not* an upper bound for a set  $X$ ? We would have to negate definition (a):  $\neg \forall x(x \leq a)$  or, equivalently,  $\exists x \neg(x \leq a)$  or  $\exists x(x > a)$ . In words,  $a$  is not an upper bound for a set  $X$  if there exists  $x$  in  $X$  such that  $x > a$ . Looking at the preceding picture, we see that any number less than  $e$  is not an upper bound for  $X$ .

Let's turn to definition (b), which says, in words, that  $a$  is a least upper bound for a set  $X$  if, among all upper bounds for  $X$ ,  $a$  is smallest. Looking ahead, problem 2 is to show that there is only one (distinct) upper bound for a set  $X$ ; thus, we usually say *the* least upper bound rather than  $a$  least upper bound. The least upper bound of our previous set  $X = \{1, 2, 3, 4\}$  is 4. We have already noted that 4 is an upper bound for  $X$ . If  $a$  is any upper bound for  $X$ , since  $4 \in X$ ,  $4 \leq a$ . Therefore, 4 is the least upper bound for  $X$ . In the preceding figure,  $e$  is a least upper bound for the set  $X$ .

### Finding a Solution

Now we consider the problems.

[Problem 1.] Our previous example,  $X = \{1, 2, 3, 4\}$ , will suffice. We have noted that 4 is the

least upper bound for  $X$ . Any values greater than 4 serve as additional upper bounds.

[Problem 2.] One way to prove that two numbers  $a$  and  $b$  are equal is to show that  $a \leq b$  and  $b \leq a$ . We'll try this first. Another possibility is proof by contradiction and assume that  $a \neq b$ .

[Problem 3.] Here we can use the fact that  $a - \varepsilon$  is *not* an upper bound (since it's less than the least upper bound) and, as discussed previously, what it means for a value to *not* be an upper bound.

[Problem 4.] Here we are given a value,  $ta$ , and asked to prove that it is the least upper bound of the given set, which here we denote as  $tX$ . Going directly to the definitions, we must show that

- (a)  $z \leq ta$  for every  $z \in tX$  (i.e.,  $ta$  is an upper bound for  $tX$ ),
- (b) if  $b$  is an upper bound for  $tX$ , then  $b \geq ta$  (i.e.,  $ta$  is the least upper bound for  $tX$ ).

For part (a), since  $z = tx$  ( $x \in X$ ), we must show that

$$tx \leq ta \quad \text{for all } x \in X.$$

We are given that  $a$  is a least upper bound for  $X$ . In particular,  $a$  is an upper bound for  $X$  so

$$x \leq a \quad \text{for all } x \in X.$$

How do we deduce the first inequality from the second? Multiply by  $t$ ! We hope that the proof of part (b) proceeds in a similar way.

### Formal Solution

[Problem 1.] Let  $X = \{1, 2, 3, 4\}$ . Upper bounds for  $X$  are 4, 5, and 6 since  $x \leq 4$ ,  $x \leq 5$ , and  $x \leq 6$  for every  $x \in X$ .

The least upper bound for  $X$  is 4. We have already noted that 4 is an upper bound for  $X$ . If  $a$  is any upper bound for  $X$ , since  $4 \in X$ ,  $4 \leq a$ . Therefore, 4 is the least upper bound for  $X$ .

[Problem 2.] Since  $a$  is a least upper bound for  $X$  and  $b$  is an upper bound for  $X$ ,  $a \leq b$ . Since  $b$  is a least upper bound for  $X$  and  $a$  is an upper bound for  $X$ ,  $b \leq a$ . Therefore,  $a = b$ .

[Problem 3.] Let  $\varepsilon > 0$ . Since  $a$  is the least upper bound for  $X$  and  $a - \varepsilon < a$ ,  $a - \varepsilon$  is *not* an upper bound for  $X$ . Therefore, by definition (a) there exists  $x \in X$  such that  $a - \varepsilon < x$ . Since  $a$  is an upper bound for  $X$ ,  $x \leq a$ . We have shown that there exists  $x \in X$  such that  $a - \varepsilon < x \leq a$ .

[Problem 4.] Let  $tX$  denote the set  $\{tx \mid x \in X\}$ . We must prove that

- (a)  $z \leq ta$  for every  $z \in tX$  (i.e.,  $ta$  is an upper bound for  $tX$ ),
- (b) if  $b$  is an upper bound for  $tX$ , then  $b \geq ta$  (i.e.,  $ta$  is the least upper bound for  $tX$ ).

We first prove part (a). Let  $z \in tX$ . Then  $z = tx$  for some  $x \in X$  (by the definition of the set  $tX$ ). Since  $a$  is an upper bound for  $X$ ,  $x \leq a$ . Multiplying by  $t$  and noting that  $t > 0$ , we have  $z = tx \leq ta$ . Therefore,  $z \leq ta$  for every  $z \in tX$  and the proof of part (a) is complete.

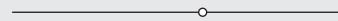
Next we prove part (b). Let  $b$  be an upper bound for  $tX$ . Then  $tx \leq b$  for every  $x \in X$  (since an arbitrary element in  $tX$  is of the form  $tx$  for some  $x \in X$ ). Dividing by  $t$  and noting that  $t > 0$ , we have  $x \leq b/t$  for every  $x \in X$ . Therefore  $b/t$  is an upper bound for  $X$ . Since  $a$  is the least upper bound for  $X$ ,  $b/t \geq a$ . Multiplying by  $t$  and noting again that  $t > 0$ , we have  $b \geq ta$ . Therefore  $ta$  is the least upper bound for  $tX$ . The proof is complete.

### Summary of Problem-Solving Techniques

- Before beginning a proof, familiarize yourself with relevant definitions, theorems, examples, and so on.
- Construct additional examples—especially small examples (e.g., for sets look at some small finite sets).
- Write out some of the technical statements in words.
- Look at negations of statements.
- Draw pictures.
- If one proof technique seems not to be working, try another. For example, if a direct proof seems unpromising, try a proof by contradiction.
- Review the Problem-Solving Tips sections in this chapter and the previous chapter.

### Comments

The fact that every nonempty set of real numbers that is bounded above has a least upper bound is called the **completeness property of the real numbers**. The real numbers are complete in the sense that there are no “holes” in the number line. Informally, if there was a hole in the line, the set of numbers to the left of the hole, although bounded above, would not have a least upper bound:



The set of rational numbers is *not* complete. The subset of rational numbers less than  $\sqrt{2}$  is bounded above, but does not have a *rational* least upper bound. (The least upper bound of the subset of rational numbers less than  $\sqrt{2}$  is the irrational number  $\sqrt{2}$ .)

## Exercises

1. What is the least upper bound of a nonempty finite set of real numbers?
2. What is the least upper bound of the set

$$\{1 - 1/n \mid n \text{ is a positive integer}\}?$$

Prove your answer.

3. Let  $X$  and  $Y$  be nonempty sets of real numbers such that  $X \subseteq Y$  and  $Y$  is bounded above. Prove that  $X$  is bounded above and  $\text{lub } X \leq \text{lub } Y$ .
4. Let  $X$  be a nonempty set. What is the least upper bound of the set  $\{tx \mid x \in X\}$  if  $t = 0$ ?
5. Let  $X$  be a set with least upper bound  $a$ , and let  $Y$  be a set with least upper bound  $b$ . Prove that the set

$$\{x + y \mid x \in X \text{ and } y \in Y\}$$

is bounded above and its least upper bound is  $a + b$ .

*Let  $X$  be a nonempty set of real numbers. A lower bound for  $X$  is a real number  $a$  having the property that  $x \geq a$  for every  $x \in X$ . Let  $a$  be a lower bound for a set  $X$  of real numbers. If every lower bound  $b$  for  $X$  satisfies  $b \leq a$ , we call  $a$  a greatest lower bound for  $X$ .*

6. Prove that if  $a$  and  $b$  are greatest lower bounds for a set  $X$ , then  $a = b$ .
7. Prove that every nonempty subset of real numbers bounded below has a greatest lower bound. *Hint:* If  $X$  is a nonempty set of real numbers bounded below, let  $Y$  denote the set of lower bounds. Prove that  $Y$  has a least upper bound, say  $a$ . Prove that  $a$  is the greatest lower bound for  $X$ .
8. Let  $X$  be a set with greatest lower bound  $a$ . Prove that if  $\varepsilon > 0$ , then there exists  $x \in X$  satisfying  $a + \varepsilon > x \geq a$ .
9. Let  $X$  be a set with least upper bound  $a$ , and let  $t < 0$ . Prove that  $ta$  is the greatest lower bound of the set  $\{tx \mid x \in X\}$ .

## 2.3 Resolution Proofs<sup>†</sup>

In this section, we will write  $a \wedge b$  as  $ab$ .

**Resolution** is a proof technique proposed by J. A. Robinson in 1965 (see [Robinson]) that depends on a single rule:

If  $p \vee q$  and  $\neg p \vee r$  are both true, then  $q \vee r$  is true. (2.3.1)

Statement (2.3.1) can be verified by writing the truth table (see Exercise 1). Because resolution depends on this single, simple rule, it is the basis of many computer programs that reason and prove theorems.

In a proof by resolution, the hypotheses and the conclusion are written as **clauses**. A clause consists of terms separated by *or's*, where each term is a variable or the negation of a variable.

### Example 2.3.1

The expression  $a \vee b \vee \neg c \vee d$  is a clause since the terms  $a$ ,  $b$ ,  $\neg c$ , and  $d$  are separated by *or's* and each term is a variable or the negation of a variable. ▲

### Example 2.3.2

The expression  $xy \vee w \vee \neg z$  is *not* a clause even though the terms are separated by *or's*, since the term  $xy$  consists of two variables—not a single variable. ▲

### Example 2.3.3

The expression  $p \rightarrow q$  is *not* a clause since the terms are separated by  $\rightarrow$ . Each term is, however, a variable. ▲

<sup>†</sup>This section can be omitted without loss of continuity.

A direct proof by resolution proceeds by repeatedly applying (2.3.1) to pairs of statements to derive new statements until the conclusion is derived. When we apply (2.3.1),  $p$  must be a single variable, but  $q$  and  $r$  can be expressions. Notice that when (2.3.1) is applied to clauses, the result  $q \vee r$  is a clause. (Since  $q$  and  $r$  each consist of terms separated by *or*'s, where each term is a variable or the negation of a variable,  $q \vee r$  also consists of terms separated by *or*'s, where each term is a variable or the negation of a variable.)

**Example 2.3.4**

Prove the following using resolution:

$$\begin{array}{l} 1. \quad a \vee b \\ 2. \quad \neg a \vee c \\ 3. \quad \underline{\neg c \vee d} \\ \therefore b \vee d \end{array}$$

**SOLUTION** Applying (2.3.1) to expressions 1 and 2, we derive

$$4. \quad b \vee c.$$

Applying (2.3.1) to expressions 3 and 4, we derive

$$5. \quad b \vee d,$$

the desired conclusion. Given the hypotheses 1, 2, and 3, we have proved the conclusion  $b \vee d$ . 

Special cases of (2.3.1) are as follows:

If  $p \vee q$  and  $\neg p$  are true, then  $q$  is true.

(2.3.2)

If  $p$  and  $\neg p \vee r$  are true, then  $r$  is true.

**Example 2.3.5**

Prove the following using resolution:

$$\begin{array}{l} 1. \quad a \\ 2. \quad \neg a \vee c \\ 3. \quad \underline{\neg c \vee d} \\ \therefore d \end{array}$$

**SOLUTION** Applying (2.3.2) to expressions 1 and 2, we derive

$$4. \quad c.$$

Applying (2.3.2) to expressions 3 and 4, we derive

$$5. \quad d,$$

the desired conclusion. Given the hypotheses 1, 2, and 3, we have proved the conclusion  $d$ . 

If a hypothesis is not a clause, it must be replaced by an equivalent expression that is either a clause or the *and* of clauses. For example, suppose that one of the hypotheses is  $\neg(a \vee b)$ . Since the negation applies to more than one term, we use the first of De Morgan's laws (see Example 1.3.11)

$$\neg(a \vee b) \equiv \neg a \neg b, \quad \neg(ab) \equiv \neg a \vee \neg b \quad (2.3.3)$$

to obtain an equivalent expression with the negation applying to single variables:  $\neg(a \vee b) \equiv \neg a \neg b$ . We then replace the original hypothesis  $\neg(a \vee b)$  by the two hypotheses  $\neg a$  and  $\neg b$ . This replacement is justified by recalling that individual hypotheses  $h_1$  and  $h_2$  are equivalent to  $h_1 h_2$  (see Definition 1.4.1 and the discussion that precedes it). Repeated use of De Morgan's laws will result in each negation applying to only one variable.

An expression that consists of terms separated by *or*'s, where each term consists of the *and* of *several* variables, may be replaced by an equivalent expression that consists of the *and* of clauses by using the equivalence

$$a \vee bc \equiv (a \vee b)(a \vee c). \quad (2.3.4)$$

In this case, we may replace the single hypothesis  $a \vee bc$  by the two hypotheses  $a \vee b$  and  $a \vee c$ . By using first De Morgan's laws (2.3.3) and then (2.3.4), we can obtain equivalent hypotheses, each of which is a clause.

### Example 2.3.6

Prove the following using resolution:

$$\begin{array}{l} 1. \quad a \vee \neg bc \\ 2. \quad \neg(a \vee d) \\ \hline \therefore \neg b \end{array}$$

**SOLUTION** We use (2.3.4) to replace hypothesis 1 with the two hypotheses  $a \vee \neg b$  and  $a \vee c$ . We use the first of De Morgan's laws (2.3.3) to replace hypothesis 2 with the two hypotheses  $\neg a$  and  $\neg d$ . The argument becomes

$$\begin{array}{l} 1. \quad a \vee \neg b \\ 2. \quad a \vee c \\ 3. \quad \neg a \\ 4. \quad \neg d \\ \hline \therefore \neg b \end{array}$$

Applying (2.3.1) to expressions 1 and 3, we immediately derive the conclusion  $\neg b$ . 

In automated reasoning systems, proof by resolution is combined with proof by contradiction. We write the negated conclusion as clauses and add the clauses to the hypothesis. We then repeatedly apply (2.3.1) until we derive a contradiction.

### Example 2.3.7

Give another proof of Example 2.3.4 by combining resolution with proof by contradiction.

**SOLUTION** We first negate the conclusion and use the first of De Morgan's laws (2.3.3) to obtain  $\neg(b \vee d) \equiv \neg b \neg d$ . We then add the clauses  $\neg b$  and  $\neg d$  to the hypotheses to obtain

$$\begin{array}{l} 1. \quad a \vee b \\ 2. \quad \neg a \vee c \\ 3. \quad \neg c \vee d \\ 4. \quad \neg b \\ 5. \quad \neg d \end{array}$$

Applying (2.3.1) to expressions 1 and 2, we derive

$$6. \quad b \vee c.$$

Applying (2.3.1) to expressions 3 and 6, we derive

$$7. \quad b \vee d.$$

Applying (2.3.1) to expressions 4 and 7, we derive

$$8. \quad d.$$

Now 5 and 8 combine to give a contradiction, and the proof is complete. 

It can be shown that resolution is *correct* and *refutation complete*. “Resolution is correct” means that if resolution derives a contradiction from a set of clauses, the clauses are inconsistent (i.e., the clauses are not all true). “Resolution is refutation complete” means that resolution will be able to derive a contradiction from a set of inconsistent clauses. Thus, if a conclusion follows from a set of hypotheses, resolution will be able to derive a contradiction from the hypotheses and the negation of the conclusion. Unfortunately, resolution does not tell us which clauses to combine in order to deduce the contradiction. A key challenge in automating a reasoning system is to help guide the search for clauses to combine. References on resolution and automated reasoning are [Gallier; Genesereth; and Wos].

### 2.3 Problem-Solving Tips

To construct a resolution proof, first replace any of the hypotheses or conclusion that is not a clause with one or more clauses. Then replace pairs of hypotheses of the form  $p \vee q$  and  $\neg p \vee r$  with  $q \vee r$  until deriving the conclusion. Remember that resolution can be combined with proof by contradiction.

## 2.3 Review Exercises

1. What rule of logic does proof by resolution use?
2. What is a clause?
3. Explain how a proof by resolution proceeds.

## 2.3 Exercises

1. Write a truth table that proves (2.3.1).

*Use resolution to derive each conclusion in Exercises 2–6. Hint: In Exercises 5 and 6, replace  $\rightarrow$  and  $\leftrightarrow$  with logically equivalent expressions that use or and and.*

$$2. \quad \neg p \vee q \vee r$$

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

$$3. \quad \neg p \vee r$$

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

$$\begin{array}{lll} 4. \quad \neg p \vee t & 5. \quad p \rightarrow q & 6. \quad p \leftrightarrow r \\ \neg q \vee s & \frac{p \vee q}{\therefore q} & \frac{r}{\therefore p} \\ \neg r \vee st & \hline p \vee q \vee r \vee u & \\ \hline \therefore s \vee t \vee u & & \end{array}$$

7. Use resolution and proof by contradiction to re-prove Exercises 2–6.

8. Use resolution and proof by contradiction to re-prove Example 2.3.6.

## 2.4 Mathematical Induction

### Go Online

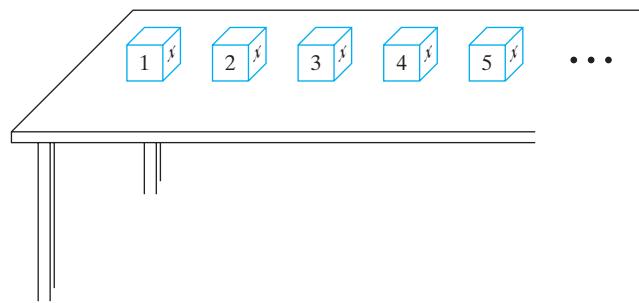
For more on mathematical induction, see [goo.gl/gHgyey](http://goo.gl/gHgyey)

Suppose that a sequence of blocks numbered 1, 2, … sits on an (infinitely) long table (see Figure 2.4.1) and that some blocks are marked with an “X.” (All of the blocks visible in Figure 2.4.1 are marked.) Suppose that

The first block is marked. (2.4.1)

For all  $n$ , if block  $n$  is marked, then block  $n + 1$  is also marked. (2.4.2)

We claim that (2.4.1) and (2.4.2) imply that every block is marked.



**Figure 2.4.1** Numbered blocks on a table.

We examine the blocks one by one. Statement (2.4.1) explicitly states that block 1 is marked. Consider block 2. Since block 1 is marked, by (2.4.2) (taking  $n = 1$ ), block 2 is also marked. Consider block 3. Since block 2 is marked, by (2.4.2) (taking  $n = 2$ ), block 3 is also marked. Continuing in this way, we can show that every block is marked. For example, suppose that we have verified that blocks 1–5 are marked, as shown in Figure 2.4.1. To show that block 6, which is not shown in Figure 2.4.1, is marked, we note that since block 5 is marked, by (2.4.2) (taking  $n = 5$ ), block 6 is also marked.

The preceding example illustrates the **Principle of Mathematical Induction**. To show how mathematical induction can be used in a more profound way, let  $S_n$  denote the sum of the first  $n$  positive integers:

$$S_n = 1 + 2 + \cdots + n. \quad (2.4.3)$$

Suppose that someone claims that

$$S_n = \frac{n(n+1)}{2} \quad \text{for all } n \geq 1. \quad (2.4.4)$$

A sequence of statements is really being made, namely,

|                                  |   |
|----------------------------------|---|
| $S_1 = \frac{1(2)}{2}$           | × |
| $S_2 = \frac{2(3)}{2}$           | × |
| ⋮                                |   |
| $S_{n-1} = \frac{(n-1)n}{2}$     | × |
| $S_n = \frac{n(n+1)}{2}$         | × |
| $S_{n+1} = \frac{(n+1)(n+2)}{2}$ | ? |
| ⋮                                |   |

$$S_1 = \frac{1(2)}{2} = 1, \quad S_2 = \frac{2(3)}{2} = 3, \quad S_3 = \frac{3(4)}{2} = 6, \dots$$

Suppose that each true equation has an “×” placed beside it (see Figure 2.4.2). Since the first equation is true, it is marked. Now suppose we can show that for all  $n$ , if equation  $n$  is marked, then equation  $n + 1$  is also marked. Then, as in the example involving the blocks, all of the equations are marked; that is, all the equations are true and the formula (2.4.4) is verified.

We must show that for all  $n$ , if equation  $n$  is true, then equation  $n + 1$  is also true. Equation  $n$  is

$$S_n = \frac{n(n+1)}{2}. \quad (2.4.5)$$

Assuming that this equation is true, we must show that equation  $n + 1$

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

is true. According to definition (2.4.3),

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

We note that  $S_n$  is contained within  $S_{n+1}$ , in the sense that

$$S_{n+1} = 1 + 2 + \cdots + n + (n + 1) = S_n + (n + 1). \quad (2.4.6)$$

**Figure 2.4.2** A sequence of statements. True statements are marked with ×.

Because of (2.4.5) and (2.4.6), we have

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

Since

$$\begin{aligned}\frac{n(n+1)}{2} + (n+1) &= \frac{n(n+1)}{2} + \frac{2(n+1)}{2} \\ &= \frac{n(n+1) + 2(n+1)}{2} \\ &= \frac{(n+1)(n+2)}{2},\end{aligned}$$

we have

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

Therefore, assuming that equation  $n$  is true, we have proved that equation  $n+1$  is true. We conclude that all of the equations are true.

Our proof using mathematical induction consisted of two steps. First, we verified that the statement corresponding to  $n = 1$  was true. Second, we *assumed* that statement  $n$  was true and then *proved* that statement  $n+1$  was also true. In proving statement  $n+1$ , we were permitted to make use of statement  $n$ ; indeed, the trick in constructing a proof using mathematical induction is to relate statement  $n$  to statement  $n+1$ .

We next formally state the Principle of Mathematical Induction.

### Principle of Mathematical Induction

Suppose that we have a propositional function  $S(n)$  whose domain of discourse is the set of positive integers. Suppose that

$$S(1) \text{ is true;} \tag{2.4.7}$$

$$\text{for all } n \geq 1, \text{ if } S(n) \text{ is true, then } S(n+1) \text{ is true.} \tag{2.4.8}$$

Then  $S(n)$  is true for every positive integer  $n$ .

Condition (2.4.7) is sometimes called the **Basis Step** and condition (2.4.8) is sometimes called the **Inductive Step**. Hereafter, “induction” will mean “mathematical induction.”

After defining  $n$  factorial, we illustrate the Principle of Mathematical Induction with another example.

**Definition 2.4.1** ►  $n$  factorial is defined as

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n(n-1)(n-2)\cdots 2 \cdot 1 & \text{if } n \geq 1. \end{cases}$$

That is, if  $n \geq 1$ ,  $n!$  is equal to the product of all the integers between 1 and  $n$  inclusive. As a special case,  $0!$  is defined to be 1. ▲

#### Example 2.4.2

$$0! = 1! = 1, \quad 3! = 3 \cdot 2 \cdot 1 = 6, \quad 6! = 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 720$$

#### Example 2.4.3

Use induction to show that

$$n! \geq 2^{n-1} \quad \text{for all } n \geq 1. \tag{2.4.9}$$

**SOLUTION****Basis Step ( $n = 1$ )**

[Condition (2.4.7)] We must show that (2.4.9) is true if  $n = 1$ . This is easily accomplished, since  $1! = 1 \geq 1 = 2^{1-1}$ .

**Inductive Step**

[Condition (2.4.8)] We assume that the inequality is true for  $n \geq 1$ ; that is, we assume that

$$n! \geq 2^{n-1} \quad (2.4.10)$$

is true. We must then prove that the inequality is true for  $n + 1$ ; that is, we must prove that

$$(n+1)! \geq 2^n \quad (2.4.11)$$

is true. We can relate (2.4.10) and (2.4.11) by observing that  $(n+1)! = (n+1)(n!)$ . Now

$$\begin{aligned} (n+1)! &= (n+1)(n!) \\ &\geq (n+1)2^{n-1} \quad \text{by (2.4.10)} \\ &\geq 2 \cdot 2^{n-1} \quad \text{since } n+1 \geq 2 \\ &= 2^n. \end{aligned}$$

Therefore, (2.4.11) is true. We have completed the Inductive Step.

Since the Basis Step and the Inductive Step have been verified, the Principle of Mathematical Induction tells us that (2.4.9) is true for every positive integer  $n$ . 

If we want to verify that the statements  $S(n_0), S(n_0 + 1), \dots$ , where  $n_0 \neq 1$ , are true, we must change the Basis Step to  $S(n_0)$  is true. In words, the Basis Step is to prove that the propositional function  $S(n)$  is true for the smallest value  $n_0$  in the domain of discourse.

The Inductive Step then becomes

*for all  $n \geq n_0$ , if  $S(n)$  is true, then  $S(n + 1)$  is true.*

**Example 2.4.4**

**Geometric Sum** Use induction to show that if  $r \neq 1$ ,

$$a + ar^1 + ar^2 + \cdots + ar^n = \frac{a(r^{n+1} - 1)}{r - 1} \quad (2.4.12)$$

for all  $n \geq 0$ .

The sum on the left is called the **geometric sum**. In the geometric sum in which  $a \neq 0$  and  $r \neq 0$ , the ratio of adjacent terms  $[(ar^{i+1})/(ar^i)] = r$  is constant.

**SOLUTION****Basis Step ( $n = 0$ )**

Since the smallest value in the domain of discourse  $\{n \mid n \geq 0\}$  is  $n = 0$ , the Basis Step is to prove that (2.4.12) is true for  $n = 0$ . For  $n = 0$ , (2.4.12) becomes

$$a = \frac{a(r^1 - 1)}{r - 1},$$

which is true.

**Inductive Step**

Assume that statement (2.4.12) is true for  $n$ . Now

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

Since the Basis Step and the Inductive Step have been verified, the Principle of Mathematical Induction tells us that (2.4.12) is true for all  $n \geq 0$ . 

As an example of the use of the geometric sum, if we take  $a = 1$  and  $r = 2$  in (2.4.12), we obtain the formula

$$1 + 2 + 2^2 + 2^3 + \cdots + 2^n = \frac{2^{n+1} - 1}{2 - 1} = 2^{n+1} - 1.$$

The reader has surely noticed that in order to prove the previous formulas, one has to be given the correct formulas in advance. A reasonable question is: How does one come up with the formulas? There are many answers to this question. One technique to derive a formula is to experiment with small values and try to discover a pattern. (Another technique is discussed in Exercises 70–73). For example, consider the sum  $1 + 3 + \cdots + (2n - 1)$ . The following table gives the values of this sum for  $n = 1, 2, 3, 4$ .

| <i>n</i> | $1 + 3 + \cdots + (2n - 1)$ |
|----------|-----------------------------|
| 1        | 1                           |
| 2        | 4                           |
| 3        | 9                           |
| 4        | 16                          |

Since the second column consists of squares, we conjecture that

$$1 + 3 + \cdots + (2n - 1) = n^2 \quad \text{for every positive integer } n.$$

The conjecture is correct and the formula can be proved by mathematical induction (see Exercise 1).

At this point, the reader may want to read the Problem-Solving Corner that follows this section. This Problem-Solving Corner gives an extended, detailed exposition of how to do proofs by mathematical induction.

Our final examples show that induction is not limited to proving formulas for sums and verifying inequalities.

**Example 2.4.5**

Use induction to show that  $5^n - 1$  is divisible by 4 for all  $n \geq 1$ .

**SOLUTION****Basis Step ( $n = 1$ )**

If  $n = 1$ ,  $5^n - 1 = 5^1 - 1 = 4$ , which is divisible by 4.

**Inductive Step**

We assume that  $5^n - 1$  is divisible by 4. We must then show that  $5^{n+1} - 1$  is divisible by 4. We use the fact that if  $p$  and  $q$  are each divisible by  $k$ , then  $p + q$  is also divisible by  $k$ . In our case,  $k = 4$ . We leave the proof of this fact to the exercises (see Exercise 74).

We relate the  $(n + 1)$ st case to the  $n$ th case by writing

$$5^{n+1} - 1 = 5^n - 1 + \text{to be determined.}$$

Now, by the inductive assumption,  $5^n - 1$  is divisible by 4. If “to be determined” is also divisible by 4, then the preceding sum, which is equal to  $5^{n+1} - 1$ , will also be divisible by 4, and the Inductive Step will be complete. We must find the value of “to be determined.”

Now

$$5^{n+1} - 1 = 5 \cdot 5^n - 1 = 4 \cdot 5^n + 1 \cdot 5^n - 1.$$

Thus, “to be determined” is  $4 \cdot 5^n$ , which is divisible by 4. Formally, we could write the Inductive Step as follows.

By the inductive assumption,  $5^n - 1$  is divisible by 4 and, since  $4 \cdot 5^n$  is divisible by 4, the sum

$$(5^n - 1) + 4 \cdot 5^n = 5^{n+1} - 1$$

is divisible by 4.

Since the Basis Step and the Inductive Step have been verified, the Principle of Mathematical Induction tells us that  $5^n - 1$  is divisible by 4 for all  $n \geq 1$ .  $\blacktriangleleft$

We next give the proof promised in Section 1.1 that if a set  $X$  has  $n$  elements, the power set of  $X$ ,  $\mathcal{P}(X)$ , has  $2^n$  elements.

### Theorem 2.4.6

If  $|X| = n$ , then

$$|\mathcal{P}(X)| = 2^n \quad (2.4.13)$$

for all  $n \geq 0$ .

**Proof** The proof is by induction on  $n$ .

#### Basis Step ( $n = 0$ )

If  $n = 0$ ,  $X$  is the empty set. The only subset of the empty set is the empty set itself; thus,

$$|\mathcal{P}(X)| = 1 = 2^0 = 2^n.$$

Thus, (2.4.13) is true for  $n = 0$ .

#### Inductive Step

Assume that (2.4.13) holds for  $n$ . Let  $X$  be a set with  $n + 1$  elements. Choose  $x \in X$ . We claim that exactly half of the subsets of  $X$  contain  $x$ , and exactly half of the subsets of  $X$  do not contain  $x$ . To see this, notice that each subset  $S$  of  $X$  that contains  $x$  can be paired uniquely with the subset obtained by removing  $x$  from  $S$  (see Figure 2.4.3). Thus exactly half of the subsets of  $X$  contain  $x$ , and exactly half of the subsets of  $X$  do not contain  $x$ .

If we let  $Y$  be the set obtained from  $X$  by removing  $x$ ,  $Y$  has  $n$  elements. By the inductive assumption,  $|\mathcal{P}(Y)| = 2^n$ . But the subsets of  $Y$  are precisely the subsets of  $X$  that do not contain  $x$ . From the argument in the preceding paragraph, we conclude that

$$|\mathcal{P}(Y)| = \frac{|\mathcal{P}(X)|}{2}.$$

Therefore,

$$|\mathcal{P}(X)| = 2|\mathcal{P}(Y)| = 2 \cdot 2^n = 2^{n+1}.$$

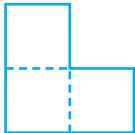
Thus (2.4.13) holds for  $n + 1$  and the inductive step is complete. By the Principle of Mathematical Induction, (2.4.13) holds for all  $n \geq 0$ .  $\blacktriangleleft$

| Subsets of $X$ that contain $a$ | Subsets of $X$ that do not contain $a$ |
|---------------------------------|----------------------------------------|
| $\{a\}$                         | $\emptyset$                            |
| $\{a, b\}$                      | $\{b\}$                                |
| $\{a, c\}$                      | $\{c\}$                                |
| $\{a, b, c\}$                   | $\{b, c\}$                             |

**Figure 2.4.3** Subsets of  $X = \{a, b, c\}$  divided into two classes: those that contain  $a$  and those that do not contain  $a$ . Each subset in the right column is obtained from the corresponding subset in the left column by deleting the element  $a$  from it.

**Example 2.4.7****Go Online**

For more on trominoes, see  
[goo.gl/gHgyey](http://goo.gl/gHgyey)



**Figure 2.4.4** A tromino.

**A Tiling Problem** A *right tromino*, hereafter called simply a *tromino*, is an object made up of three squares, as shown in Figure 2.4.4. A tromino is a type of polyomino. Since polyominoes were introduced by Solomon W. Golomb in 1954 (see [Golomb, 1954]), they have been a favorite topic in recreational mathematics. A *Polyomino of order s* consists of  $s$  squares joined at the edges. A tromino is a polyomino of order 3. Three squares in a row form the only other type of polyomino of order 3. (No one has yet found a simple formula for the number of polyominoes of order  $s$ .) Numerous problems using polyominoes have been devised (see [Martin]).

We give Golomb's inductive proof (see [Golomb, 1954]) that if we remove one square from an  $n \times n$  board, where  $n$  is a power of 2, we can tile the remaining squares with right trominoes (see Figure 2.4.5). By a *tiling* of a figure by trominoes, we mean an exact covering of the figure by trominoes without having any of the trominoes overlap each other or extend outside the figure. We call a board with one square missing a *deficient board*.

We now use induction on  $k$  to prove that we can tile a  $2^k \times 2^k$  deficient board with trominoes for all  $k \geq 1$ .

**Basis Step ( $k = 1$ )**

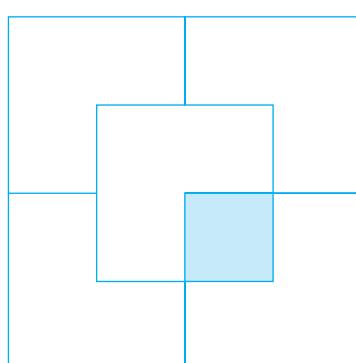
If  $k = 1$ , the  $2 \times 2$  deficient board is itself a tromino and can therefore be tiled with one tromino.

**Inductive Step**

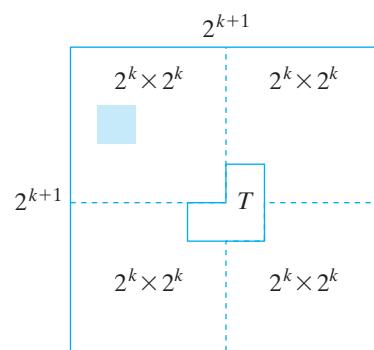
Assume that we can tile a  $2^k \times 2^k$  deficient board. We show that we can tile a  $2^{k+1} \times 2^{k+1}$  deficient board.

Consider a  $2^{k+1} \times 2^{k+1}$  deficient board. Divide the board into four  $2^k \times 2^k$  boards, as shown in Figure 2.4.6. Rotate the board so that the missing square is in the upper-left quadrant. By the inductive assumption, the upper-left  $2^k \times 2^k$  board can be tiled. Place one tromino  $T$  in the center, as shown in Figure 2.4.6, so that each square of  $T$  is in each of the other quadrants. If we consider the squares covered by  $T$  as missing, each of these quadrants is a  $2^k \times 2^k$  deficient board. Again, by the inductive assumption, these boards can be tiled. We now have a tiling of the  $2^{k+1} \times 2^{k+1}$  board. By the Principle of Mathematical Induction, it follows that any  $2^k \times 2^k$  deficient board can be tiled with trominoes,  $k = 1, 2, \dots$ .

If we can tile an  $n \times n$  deficient board, where  $n$  is not necessarily a power of 2, then the number of squares,  $n^2 - 1$ , must be divisible by 3. [Chu] showed that the converse is true, except when  $n$  is 5. More precisely, if  $n \neq 5$ , any  $n \times n$  deficient board can be tiled



**Figure 2.4.5** Tiling a  $4 \times 4$  deficient board with trominoes.



**Figure 2.4.6** Using mathematical induction to tile a  $2^{k+1} \times 2^{k+1}$  deficient board with trominoes.

with trominoes if and only if 3 divides  $n^2 - 1$  (see Exercises 28 and 29, Section 2.5). [Some  $5 \times 5$  deficient boards can be tiled and some cannot (see Exercises 33–35).]

Some real-world problems can be modeled as tiling problems. One example is the *VLSI layout problem*—the problem of packing many components on a computer chip (see [Wong]). (VLSI is short for very large scale integration.) The problem is to tile a rectangle of minimum area with the desired components. The components are sometimes modeled as rectangles and L-shaped figures similar to (right) trominoes. In practice, other constraints are imposed such as the proximity of various components that must be interconnected and restrictions on the ratios of width to height of the resulting rectangle. ▲

A **loop invariant** is a statement about program variables that is true just before a loop begins executing and is also true after each iteration of the loop. In particular, a loop invariant is true after the loop finishes, at which point the invariant tells us something about the state of the variables. Ideally, this statement tells us that the loop produces the expected result, that is, that the loop is correct. For example, a loop invariant for a while loop

```
while (condition)
 // loop body
```

is true just before *condition* is evaluated the first time, and it is also true each time the loop body is executed.

We can use mathematical induction to prove that an invariant has the desired behavior. The Basis Step proves that the invariant is true before the condition that controls looping is tested for the first time. The Inductive Step assumes that the invariant is true and then proves that if the condition that controls looping is true (so that the loop body is executed again), the invariant is true after the loop body executes. Since a loop iterates a finite number of times, the form of mathematical induction used here proves that a *finite* sequence of statements is true, rather than an infinite sequence of statements as in our previous examples. Whether the sequence of statements is finite or infinite, the steps needed for the proof by mathematical induction are the same. We illustrate a loop invariant with an example.

### Example 2.4.8

Use a loop invariant to prove that when the pseudocode

```
i = 1
fact = 1
while (i < n) {
 i = i + 1
 fact = fact * i
}
```

terminates, *fact* is equal to  $n!$ .

**SOLUTION** We prove that  $fact = i!$  is an invariant for the while loop. Just before the while loop begins executing,  $i = 1$  and  $fact = 1$ , so  $fact = 1!$ . We have proved the Basis Step.

Assume that  $fact = i!$ . If  $i < n$  is true (so that the loop body executes again),  $i$  becomes  $i + 1$  and  $fact$  becomes

$$fact * (i + 1) = i! * (i + 1) = (i + 1)!.$$

We have proved the Inductive Step. Therefore,  $fact = i!$  is an invariant for the while loop.

The while loop terminates when  $i = n$ . Because  $\text{fact} = i!$  is an invariant, at this point,  $\text{fact} = n!$ .

## 2.4 Problem-Solving Tips

To prove

$$a_1 + a_2 + \cdots + a_n = F(n) \quad \text{for all } n \geq 1,$$

where  $F(n)$  is the formula for the sum, first verify the equation for  $n = 1$ :  $a_1 = F(1)$  (Basis Step). This is usually straightforward.

Now assume that the statement is true for  $n$ ; that is, assume

$$a_1 + a_2 + \cdots + a_n = F(n).$$

Add  $a_{n+1}$  to both sides to get

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

Finally, show that

$$F(n) + a_{n+1} = F(n+1).$$

To verify the preceding equation, use algebra to manipulate the left-hand side of the equation [ $F(n) + a_{n+1}$ ] until you get  $F(n+1)$ . *Look at  $F(n+1)$  so you know where you're headed.* (It's somewhat like looking up the answer in the back of the book!) You've shown that

$$a_1 + a_2 + \cdots + a_{n+1} = F(n+1),$$

which is the Inductive Step. Now the proof is complete.

Proving an inequality is handled in a similar fashion. The difference is that instead of obtaining equality [ $F(n) + a_{n+1} = F(n+1)$  in the preceding discussion], you obtain an *inequality*.

In general, the key to devising a proof by induction is to find case  $n$  “within” case  $n+1$ . Review the tiling problem (Example 2.4.7), which provides a striking example of case  $n$  “within” case  $n+1$ .

## 2.4 Review Exercises

1. State the Principle of Mathematical Induction.
2. Explain how a proof by mathematical induction proceeds.
3. Give a formula for the sum  $1 + 2 + \cdots + n$ .
4. What is the geometric sum? Give a formula for it.

## 2.4 Exercises

*In Exercises 1–12, using induction, verify that each equation is true for every positive integer  $n$ .*

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

$$2. 1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 + \cdots + n(n+1) = \frac{n(n+1)(n+2)}{3}$$

$$3. 1(1!) + 2(2!) + \cdots + n(n!) = (n+1)! - 1$$

$$4. 1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

$$5. 1^2 - 2^2 + 3^2 - \cdots + (-1)^{n+1} n^2 = \frac{(-1)^{n+1} n(n+1)}{2}$$

$$6. 1^3 + 2^3 + 3^3 + \cdots + n^3 = \left[ \frac{n(n+1)}{2} \right]^2$$

$$7. \frac{1}{1 \cdot 3} + \frac{1}{3 \cdot 5} + \frac{1}{5 \cdot 7} + \cdots + \frac{1}{(2n-1)(2n+1)} = \frac{n}{2n+1}$$

$$\begin{aligned} 8. \quad & \frac{1}{2 \cdot 4} + \frac{1 \cdot 3}{2 \cdot 4 \cdot 6} + \frac{1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6 \cdot 8} + \cdots + \frac{1 \cdot 3 \cdot 5 \cdots (2n-1)}{2 \cdot 4 \cdot 6 \cdots (2n+2)} \\ &= \frac{1}{2} - \frac{1 \cdot 3 \cdot 5 \cdots (2n+1)}{2 \cdot 4 \cdot 6 \cdots (2n+2)} \end{aligned}$$

$$\begin{aligned} 9. \quad & \frac{1}{2^2 - 1} + \frac{1}{3^2 - 1} + \cdots + \frac{1}{(n+1)^2 - 1} \\ &= \frac{3}{4} - \frac{1}{2(n+1)} - \frac{1}{2(n+2)} \end{aligned}$$

$$10. \quad 1 \cdot 2^2 + 2 \cdot 3^2 + \cdots + n(n+1)^2 = \frac{n(n+1)(n+2)(3n+5)}{12}$$

$$11. \quad \cos x + \cos 2x + \cdots + \cos nx = \frac{\cos[(x/2)(n+1)] \sin(nx/2)}{\sin(x/2)}$$

provided that  $\sin(x/2) \neq 0$ .

$$12. \quad 1 \sin x + 2 \sin 2x + \cdots + n \sin nx$$

$$= \frac{\sin[(n+1)x]}{4 \sin^2(x/2)} - \frac{(n+1) \cos[(2n+1)x/2]}{2 \sin(x/2)}$$

provided that  $\sin(x/2) \neq 0$ .

*In Exercises 13–18, using induction, verify the inequality.*

$$13. \quad \frac{1}{2n} \leq \frac{1 \cdot 3 \cdot 5 \cdots (2n-1)}{2 \cdot 4 \cdot 6 \cdots (2n)}, \quad n = 1, 2, \dots$$

$$14. \quad \frac{1 \cdot 3 \cdot 5 \cdots (2n-1)}{2 \cdot 4 \cdot 6 \cdots (2n)} \leq \frac{1}{\sqrt{n+1}}, \quad n = 1, 2, \dots$$

$$15. \quad 2n+1 \leq 2^n, \quad n = 3, 4, \dots$$

$$16. \quad 2^n \geq n^2, \quad n = 4, 5, \dots$$

$$17. \quad (a_1 a_2 \cdots a_{2^n})^{1/2^n} \leq \frac{a_1 + a_2 + \cdots + a_{2^n}}{2^n}, \quad n = 1, 2, \dots, \text{ and the } a_i \text{ are positive numbers}$$

$$18. \quad (1+x)^n \geq 1+nx, \text{ for } x \geq -1 \text{ and } n \geq 1$$

19. Use the geometric sum to prove that

$$r^0 + r^1 + \cdots + r^n < \frac{1}{1-r}$$

for all  $n \geq 0$  and  $0 < r < 1$ .

20. Prove that

$$1 \cdot r^1 + 2 \cdot r^2 + \cdots + nr^n < \frac{r}{(1-r)^2}$$

for all  $n \geq 1$  and  $0 < r < 1$ . Hint: Using the result of the previous exercise, compare the sum of the terms in

$$\begin{array}{ccccccc} r & r^2 & r^3 & r^4 & \cdots & r^n \\ r^2 & r^3 & r^4 & \cdots & & r^n \\ r^3 & r^4 & \cdots & & & r^n \\ r^4 & \cdots & & & & \\ \vdots & \vdots & & & & \\ r^{n-1} & r^n & & & & \\ r^n & & & & & \end{array}$$

in the diagonal direction ( $\swarrow$ ) with the sum of the terms by columns.

21. Prove that

$$\frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots + \frac{n}{2^n} < 2$$

for all  $n \geq 1$ .

*In Exercises 22–25, use induction to prove the statement.*

$$22. \quad 7^n - 1 \text{ is divisible by 6, for all } n \geq 1.$$

$$23. \quad 11^n - 6 \text{ is divisible by 5, for all } n \geq 1.$$

$$24. \quad 6 \cdot 7^n - 2 \cdot 3^n \text{ is divisible by 4, for all } n \geq 1.$$

$$25. \quad 3^n + 7^n - 2 \text{ is divisible by 8, for all } n \geq 1.$$

26. Use induction to prove that if  $X_1, \dots, X_n$  and  $X$  are sets, then

$$(a) \quad X \cap (X_1 \cup X_2 \cup \cdots \cup X_n) = (X \cap X_1) \cup (X \cap X_2) \cup \cdots \cup (X \cap X_n).$$

$$(b) \quad \overline{X_1 \cap X_2 \cap \cdots \cap X_n} = \overline{X_1} \cup \overline{X_2} \cup \cdots \cup \overline{X_n}.$$

27. Use induction to prove that if  $X_1, \dots, X_n$  are sets, then

$$|X_1 \times X_2 \times \cdots \times X_n| = |X_1| \cdot |X_2| \cdots |X_n|.$$

28. Prove that the number of subsets  $S$  of  $\{1, 2, \dots, n\}$ , with  $|S|$  even, is  $2^{n-1}$ ,  $n \geq 1$ .

29. By experimenting with small values of  $n$ , guess a formula for the given sum,

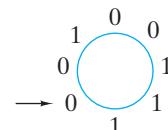
$$\frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \cdots + \frac{1}{n(n+1)};$$

then use induction to verify your formula.

30. Use induction to show that  $n$  straight lines in the plane divide the plane into  $(n^2 + n + 2)/2$  regions. Assume that no two lines are parallel and that no three lines have a common point.

31. Show that the regions of the preceding exercise can be colored red and green so that no two regions that share an edge are the same color.

32. Given  $n$  0's and  $n$  1's distributed in any manner whatsoever around a circle (see the following figure), show, using induction on  $n$ , that it is possible to start at some number and proceed clockwise around the circle to the original starting position so that, at any point during the cycle, we have seen at least as many 0's as 1's. In the following figure, a possible starting point is marked with an arrow.



33. Give a tiling of a  $5 \times 5$  board with trominoes in which the upper-left square is missing.

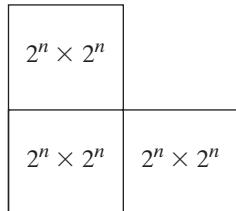
34. Show a  $5 \times 5$  deficient board that is impossible to tile with trominoes. Explain why your board cannot be tiled with trominoes.

35. Which  $5 \times 5$  deficient boards can be tiled?

36. Show that any  $(2i) \times (3j)$  board, where  $i$  and  $j$  are positive integers, with no square missing, can be tiled with trominoes.

37. Show that any  $7 \times 7$  deficient board can be tiled with trominoes.

38. Show that any  $11 \times 11$  deficient board can be tiled with trominoes. Hint: Subdivide the board into overlapping  $7 \times 7$  and  $5 \times 5$  boards and two  $6 \times 4$  boards. Then, use Exercises 33, 36, and 37.
39. This exercise and the one that follows are due to Anthony Quas. A  $2^n \times 2^n$  L-shape,  $n \geq 0$ , is a figure of the form



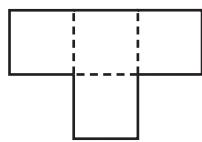
with no missing squares. Show that any  $2^n \times 2^n$  L-shape can be tiled with trominoes.

40. Use the preceding exercise to give a different proof that any  $2^n \times 2^n$  deficient board can be tiled with trominoes.

A straight tromino is an object made up of three squares in a row:



41. Which  $4 \times 4$  deficient boards can be tiled with straight trominoes? Hint: Number the squares of the  $4 \times 4$  board, left to right, top to bottom: 1, 2, 3, 1, 2, 3, and so on. Note that if there is a tiling, each straight tromino covers exactly one 2 and exactly one 3.
42. Which  $5 \times 5$  deficient boards can be tiled with straight trominoes?
43. Which  $8 \times 8$  deficient boards can be tiled with straight trominoes?
- \*44. A T-tetromino is an object made up of four squares



Prove that an  $m \times n$  rectangle can be tiled with T-tetrominoes if and only if 4 divides  $m$  and 4 divides  $n$ .

45. Use a loop invariant to prove that when the pseudocode

```
i = 1
pow = 1
while (i ≤ n) {
 pow = pow * a
 i = i + 1
}
```

terminates,  $pow$  is equal to  $a^n$ .

46. Prove that, after the following pseudocode terminates,  $a[h] = val$ ; for all  $p$ ,  $i \leq p < h$ ,  $a[p] < val$ ; and for all  $p$ ,  $h < p \leq j$ ,  $a[p] \geq val$ . In particular,  $val$  is in the position in the array  $a[i], \dots, a[j]$  where it would be if the array were sorted.

```
val = a[i]
h = i
for k = i + 1 to j
 if (a[k] < val) {
 h = h + 1
 swap(a[h], a[k])
 }
swap(a[i], a[h])
```

Hint: Use the loop invariant:  $h < k$ ; for all  $p$ ,  $i < p \leq h$ ,  $a[p] < val$ ; and, for all  $p$ ,  $h < p < k$ ,  $a[p] \geq val$ . (A picture is helpful.)

This technique is called *partitioning*. This particular version is due to Nico Lomuto. Partitioning can be used to find the  $k$ th smallest element in an array and to construct a sorting algorithm called *quicksort*.

A 3D-septomino is a three-dimensional  $2 \times 2 \times 2$  cube with one  $1 \times 1 \times 1$  corner cube removed. A deficient cube is a  $k \times k \times k$  cube with one  $1 \times 1 \times 1$  cube removed.

47. Prove that a  $2^n \times 2^n \times 2^n$  deficient cube can be tiled by 3D-septominoes.
48. Prove that if a  $k \times k \times k$  deficient cube can be tiled by 3D-septominoes, then 7 divides one of  $k - 1, k - 2, k - 4$ .
49. Suppose that  $S_n = (n+2)(n-1)$  is (incorrectly) proposed as a formula for

$$2 + 4 + \dots + 2n.$$

- (a) Show that the Inductive Step is satisfied but that the Basis Step fails.
- ★(b) If  $S'_n$  is an arbitrary expression that satisfies the Inductive Step, what form must  $S'_n$  assume?

- \*50. What is wrong with the following argument, which allegedly shows that any two positive integers are equal?

We use induction on  $n$  to “prove” that if  $a$  and  $b$  are positive integers and  $n = \max\{a, b\}$ , then  $a = b$ .

### Basis Step ( $n=1$ )

If  $a$  and  $b$  are positive integers and  $1 = \max\{a, b\}$ , we must have  $a = b = 1$ .

### Inductive Step

Assume that if  $a'$  and  $b'$  are positive integers and  $n = \max\{a', b'\}$ , then  $a' = b'$ . Suppose that  $a$  and  $b$  are positive integers and that  $n + 1 = \max\{a, b\}$ . Now  $n = \max\{a - 1, b - 1\}$ . By the inductive hypothesis,  $a - 1 = b - 1$ . Therefore,  $a = b$ .

Since we have verified the Basis Step and the Inductive Step, by the Principle of Mathematical Induction, any two positive integers are equal!

51. What is wrong with the following “proof” that

$$\frac{1}{2} + \frac{2}{3} + \dots + \frac{n}{n+1} \neq \frac{n^2}{n+1}$$

for all  $n \geq 2$ ?

Suppose by way of contradiction that

$$\frac{1}{2} + \frac{2}{3} + \cdots + \frac{n}{n+1} = \frac{n^2}{n+1}. \quad (2.4.14)$$

Then also

$$\frac{1}{2} + \frac{2}{3} + \cdots + \frac{n}{n+1} + \frac{n+1}{n+2} = \frac{(n+1)^2}{n+2}.$$

We could prove statement (2.4.14) by induction. In particular, the Inductive Step would give

$$\left( \frac{1}{2} + \frac{2}{3} + \cdots + \frac{n}{n+1} \right) + \frac{n+1}{n+2} = \frac{n^2}{n+1} + \frac{n+1}{n+2}.$$

Therefore,

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

Multiplying each side of this last equation by  $(n+1)(n+2)$  gives

$$n^2(n+2) + (n+1)^2 = (n+1)^3.$$

This last equation can be rewritten as

$$n^3 + 2n^2 + n^2 + 2n + 1 = n^3 + 3n^2 + 3n + 1$$

or

$$n^3 + 3n^2 + 2n + 1 = n^3 + 3n^2 + 3n + 1,$$

which is a contradiction. Therefore,

$$\frac{1}{2} + \frac{2}{3} + \cdots + \frac{n}{n+1} \neq \frac{n^2}{n+1},$$

as claimed.

52. Use mathematical induction to prove that

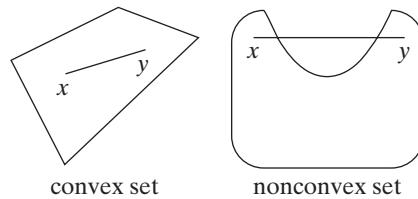
$$\frac{1}{2} + \frac{2}{3} + \cdots + \frac{n}{n+1} < \frac{n^2}{n+1}$$

for all  $n \geq 2$ . This inequality gives a correct proof of the statement of the preceding exercise.

*In Exercises 53–57, suppose that  $n > 1$  people are positioned in a field (Euclidean plane) so that each has a unique nearest neighbor. Suppose further that each person has a pie that is hurled at the nearest neighbor. A survivor is a person that is not hit by a pie.*

53. Give an example to show that if  $n$  is even, there might be no survivor.  
 54. Give an example to show that there might be more than one survivor.  
 55. [Carmony] Use induction on  $n$  to show that if  $n$  is odd, there is always at least one survivor.  
 56. Prove or disprove: If  $n$  is odd, one of two persons farthest apart is a survivor.  
 57. Prove or disprove: If  $n$  is odd, a person who throws a pie the greatest distance is a survivor.

*Exercises 58–61 deal with plane convex sets. A plane convex set, subsequently abbreviated to “convex set,” is a nonempty set  $X$  in the plane having the property that if  $x$  and  $y$  are any two points in  $X$ , the straight-line segment from  $x$  to  $y$  is also in  $X$ . The following figures illustrate.*



58. Prove that if  $X$  and  $Y$  are convex sets and  $X \cap Y$  is nonempty,  $X \cap Y$  is a convex set.  
 59. Suppose that  $X_1, X_2, X_3, X_4$  are convex sets, each three of which have a common point. Prove that all four sets have a common point.  
 60. Prove *Helly's Theorem*: Suppose that  $X_1, X_2, \dots, X_n$ ,  $n \geq 4$ , are convex sets, each three of which have a common point. Prove that all  $n$  sets have a common point.  
 61. Suppose that  $n \geq 3$  points in the plane have the property that each three of them are contained in a circle of radius 1. Prove that there is a circle of radius 1 that contains all of the points.  
 62. If  $a$  and  $b$  are real numbers with  $a < b$ , an *open interval*  $(a, b)$  is the set of all real numbers  $x$  such that  $a < x < b$ . Prove that if  $I_1, \dots, I_n$  is a set of  $n \geq 2$  open intervals such that each pair has a nonempty intersection, then

$$I_1 \cap I_2 \cap \cdots \cap I_n$$

is nonempty.

*Flavius Josephus was a Jewish soldier and historian who lived in the first century (see [Graham, 1994; Schumer]). He was one of the leaders of a Jewish revolt against Rome in the year 66. The following year, he was among a group of trapped soldiers who decided to commit suicide rather than be captured. One version of the story is that, rather than being captured, they formed a circle and proceeded around the circle killing every third person. Josephus, being proficient in discrete math, figured out where he and a buddy should stand so they could avoid being killed.*

*Exercises 63–69 concern a variant of the Josephus Problem in which every second person is eliminated. We assume that  $n$  people are arranged in a circle and numbered 1, 2, ...,  $n$  clockwise. Then, proceeding clockwise, 2 is eliminated, 4 is eliminated, and so on, until there is one survivor, denoted  $J(n)$ .*

63. Compute  $J(4)$ .  
 64. Compute  $J(6)$ .  
 65. Compute  $J(10)$ .  
 66. Use induction to show that  $J(2^i) = 1$  for all  $i \geq 1$ .  
 67. Given a value of  $n \geq 2$ , let  $2^i$  be the greatest power of 2 with  $2^i \leq n$ . (Examples: If  $n = 10$ ,  $i = 3$ . If  $n = 16$ ,  $i = 4$ .) Let  $j = n - 2^i$ . (After subtracting  $2^i$ , the greatest power of 2 less than or equal to  $n$ , from  $n$ ,  $j$  is what is left over.) By using the result of Exercise 66 or otherwise, prove that

$$J(n) = 2j + 1.$$

68. Use the result of Exercise 67 to compute  $J(1000)$ .  
 69. Use the result of Exercise 67 to compute  $J(100,000)$ .

If  $a_1, a_2, \dots$  is a sequence, we define the difference operator  $\Delta$  to be

$$\Delta a_n = a_{n+1} - a_n.$$

The formula of Exercise 70 can sometimes be used to find a formula for a sum as opposed to using induction to prove a formula for a sum (see Exercises 71–73).

- 70.** Suppose that  $\Delta a_n = b_n$ . Show that

$$b_1 + b_2 + \cdots + b_n = a_{n+1} - a_1.$$

This formula is analogous to the calculus formula  $\int_c^d f(x) dx = g(d) - g(c)$ , where  $Dg = f$  ( $D$  is the derivative operator). In the calculus formula, sum is replaced by integral, and  $\Delta$  is replaced by derivative.

## Problem-Solving Corner

### Problem

Define

$$H_k = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k} \quad (1)$$

for all  $k \geq 1$ . The numbers  $H_1, H_2, \dots$  are called the *harmonic numbers*. Prove that

$$H_{2^n} \geq 1 + \frac{n}{2} \quad (2)$$

for all  $n \geq 0$ .

### Attacking the Problem

It's often a good idea to begin attacking a problem by looking at some concrete examples of the expressions under consideration. Let's look at  $H_k$  for some small values of  $k$ . The smallest value of  $k$  for which  $H_k$  is defined is  $k = 1$ . In this case, the last term  $1/k$  in the definition of  $H_k$  equals  $1/1 = 1$ . Since the first and last terms coincide,  $H_1 = 1$ . For  $k = 2$ , the last term  $1/k$  in the definition of  $H_k$  equals  $1/2$ , so

$$H_2 = 1 + \frac{1}{2}.$$

Similarly, we find that

$$H_3 = 1 + \frac{1}{2} + \frac{1}{3},$$

$$H_4 = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4}.$$

We observe that  $H_1$  appears as the first term of  $H_2, H_3$ , and  $H_4$ , that  $H_2$  appears as the first two terms of  $H_3$  and  $H_4$ , and that  $H_3$  appears as the first three terms of  $H_4$ . In general,  $H_m$  appears as the first  $m$  terms of

- 71.** Let  $a_n = n^2$ , and compute  $\Delta a_n$ . Use Exercise 70 to find a formula for

$$1 + 2 + 3 + \cdots + n.$$

- 72.** Use Exercise 70 to find a formula for

$$1(1!) + 2(2!) + \cdots + n(n!).$$

(Compare with Exercise 3.)

- 73.** Use Exercise 70 to find a formula for

$$\frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \cdots + \frac{1}{n(n+1)}.$$

(Compare with Exercise 29.)

- 74.** Prove that if  $p$  and  $q$  are divisible by  $k$ , then  $p + q$  is divisible by  $k$ .

## Mathematical Induction

$H_k$  if  $m \leq k$ . This observation will help us later because the Inductive Step in a proof by induction must relate smaller instances of a problem to larger instances of the problem.

In general, it's a good strategy to delay combining terms and simplifying until as late as possible, which is why, for example, we left  $H_4$  as the sum of four terms rather than writing  $H_4 = 25/12$ . Since we left  $H_4$  as the sum of four terms, we were able to see that each of  $H_1, H_2$ , and  $H_3$  appears in the expression for  $H_4$ .

### Finding a Solution

The Basis Step is to prove the given statement for the smallest value of  $n$ , which here is  $n = 0$ . For  $n = 0$ , inequality (2) that we must prove becomes

$$H_{2^0} \geq 1 + \frac{0}{2} = 1.$$

We have already observed that  $H_1 = 1$ . Thus inequality (2) is true when  $n = 0$ ; in fact, the inequality is an equality. (Recall that by definition, if  $x = y$  is true, then  $x \geq y$  is also true.)

Let's move to the Inductive Step. It's a good idea to write down what is assumed (here the case  $n$ ),

$$H_{2^n} \geq 1 + \frac{n}{2}, \quad (3)$$

and what needs to be proved (here the case  $n + 1$ ),

$$H_{2^{n+1}} \geq 1 + \frac{n+1}{2}. \quad (4)$$

It's also a good idea to write the formulas for any expressions that occur. Using equation (1), we may write

$$H_{2^n} = 1 + \frac{1}{2} + \cdots + \frac{1}{2^n} \quad (5)$$

and

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

It's not so evident from the last equation that  $H_{2^n}$  appears as the first  $2^n$  terms of  $H_{2^{n+1}}$ . Let's rewrite the last equation as

$$H_{2^{n+1}} = 1 + \frac{1}{2} + \cdots + \frac{1}{2^n} + \frac{1}{2^n + 1} + \cdots + \frac{1}{2^{n+1}} \quad (6)$$

to make it clear that  $H_{2^n}$  appears as the first  $2^n$  terms of  $H_{2^{n+1}}$ .

For clarity, we have written the term that follows  $1/2^n$ . Notice that the denominators increase by one, so the term that follows  $1/2^n$  is  $1/(2^n + 1)$ . Also notice that there is a big difference between  $1/(2^n + 1)$ , the term that follows  $1/2^n$ , and  $1/2^{n+1}$ , the last term in equation (6).

Using equations (5) and (6), we may relate  $H_{2^n}$  to  $H_{2^{n+1}}$  explicitly by writing

$$H_{2^{n+1}} = H_{2^n} + \frac{1}{2^n + 1} + \cdots + \frac{1}{2^{n+1}}. \quad (7)$$

Combining (3) and (7), we obtain

$$H_{2^{n+1}} \geq 1 + \frac{n}{2} + \frac{1}{2^n + 1} + \cdots + \frac{1}{2^{n+1}}. \quad (8)$$

This inequality shows that  $H_{2^{n+1}}$  is greater than or equal to

$$1 + \frac{n}{2} + \frac{1}{2^n + 1} + \cdots + \frac{1}{2^{n+1}},$$

but our goal (4) is to show that  $H_{2^{n+1}}$  is greater than or equal to  $1 + (n + 1)/2$ . We will achieve our goal if we show that

$$1 + \frac{n}{2} + \frac{1}{2^n + 1} + \cdots + \frac{1}{2^{n+1}} \geq 1 + \frac{n+1}{2}.$$

In general, to prove an inequality, we replace terms in the larger expression with smaller terms so that the resulting expression equals the smaller expression; or we replace terms in the smaller expression with larger terms so that the resulting expression equals the larger expression. Here let's replace each of the terms in the sum

$$\frac{1}{2^n + 1} + \cdots + \frac{1}{2^{n+1}}$$

by the smallest term  $1/2^{n+1}$  in the sum. We obtain

$$\frac{1}{2^n + 1} + \cdots + \frac{1}{2^{n+1}} \geq \frac{1}{2^{n+1}} + \cdots + \frac{1}{2^{n+1}}.$$

Since there are  $2^n$  terms in the latter sum, each equal to  $1/2^{n+1}$ , we may rewrite the preceding inequality as

$$\begin{aligned} \frac{1}{2^n + 1} + \cdots + \frac{1}{2^{n+1}} &\geq \frac{1}{2^{n+1}} + \cdots + \frac{1}{2^{n+1}} \\ &= 2^n \frac{1}{2^{n+1}} = \frac{1}{2}. \end{aligned} \quad (9)$$

Combining (8) and (9),

$$H_{2^{n+1}} \geq 1 + \frac{n}{2} + \frac{1}{2} = 1 + \frac{n+1}{2}.$$

We have the desired result, and the Inductive Step is complete.

## Formal Solution

The formal solution could be written as follows.

### Basis Step ( $n = 0$ )

$$H_{2^0} = 1 \geq 1 = 1 + \frac{0}{2}$$

### Inductive Step

We assume (2). Now

$$\begin{aligned} H_{2^{n+1}} &= 1 + \frac{1}{2} + \cdots + \frac{1}{2^n} + \frac{1}{2^n + 1} + \cdots + \frac{1}{2^{n+1}} \\ &= H_{2^n} + \frac{1}{2^n + 1} + \cdots + \frac{1}{2^{n+1}} \\ &\geq 1 + \frac{n}{2} + \frac{1}{2^{n+1}} + \cdots + \frac{1}{2^{n+1}} \\ &= 1 + \frac{n}{2} + 2^n \frac{1}{2^{n+1}} \\ &= 1 + \frac{n}{2} + \frac{1}{2} = 1 + \frac{n+1}{2}. \end{aligned}$$

## Summary of Problem-Solving Techniques

- Look at concrete examples of the expressions under consideration, typically for small values of the variables.
- Look for expressions for small values of  $n$  to appear within expressions for larger values of  $n$ . In particular, the Inductive Step depends on relating case  $n$  to case  $n + 1$ .
- Delay combining terms and simplifying until as late as possible to help discover relationships among the expressions.
- Write out in full the specific cases to prove, specifically, the smallest value of  $n$  for the Basis Step, the case  $n$  that is assumed in the Inductive Step, and the case  $n + 1$  to prove in the Inductive Step. Write out the formulas for the various expressions that appear.

- To prove an inequality, replace terms in the larger expression with smaller terms so that the resulting expression equals the smaller expression, or replace terms in the smaller expression with larger terms so that the resulting expression equals the larger expression.

### Comments

The series

$$1 + \frac{1}{2} + \frac{1}{3} + \dots,$$

which surfaces in calculus, is called the *harmonic series*. Inequality (2) shows that the harmonic numbers increase without bound. In calculus terminology, the harmonic series *diverges*.

### Exercises

- Prove that  $H_{2^n} \leq 1 + n$  for all  $n \geq 0$ .

- Prove that

$$H_1 + H_2 + \dots + H_n = (n+1)H_n - n$$

for all  $n \geq 1$ .

- Prove that

$$H_n = H_{n+1} - \frac{1}{n+1}$$

for all  $n \geq 1$ .

- Prove that

$$\begin{aligned} 1 \cdot H_1 + 2 \cdot H_2 + \dots + nH_n \\ = \frac{n(n+1)}{2} H_{n+1} - \frac{n(n+1)}{4} \end{aligned}$$

for all  $n \geq 1$ .

- Prove that

$$\frac{H_1}{1} + \frac{H_2}{2} + \dots + \frac{H_n}{n} = \frac{H_n^2}{2} + \frac{1}{2} \left[ \frac{1}{1^2} + \frac{1}{2^2} + \dots + \frac{1}{n^2} \right]$$

for all  $n \geq 1$ .

## 2.5 Strong Form of Induction and the Well-Ordering Property

In the Inductive Step of mathematical induction presented in Section 2.4, we assume that statement  $n$  is true, and then prove that statement  $n+1$  is true. In other words, to prove that a statement is true (statement  $n+1$ ), we assume the truth of its immediate predecessor (statement  $n$ ). In some cases in the Inductive Step, to prove a statement is true, it is helpful to assume the truth of *all* of the preceding statements (not just the immediate predecessor). The **Strong Form of Mathematical Induction** allows us to assume the truth of all of the preceding statements. Following the usual convention, the statement to prove is denoted  $n$  rather than  $n+1$ . We next formally state the Strong Form of Mathematical Induction.

### Strong Form of Mathematical Induction

Suppose that we have a propositional function  $S(n)$  whose domain of discourse is the set of integers greater than or equal to  $n_0$ . Suppose that

$S(n_0)$  is true;

for all  $n > n_0$ , if  $S(k)$  is true for all  $k$ ,  $n_0 \leq k < n$ , then  $S(n)$  is true.

Then  $S(n)$  is true for every integer  $n \geq n_0$ .

In the Inductive Step of the Strong Form of Mathematical Induction, we let  $n$  denote an arbitrary integer,  $n > n_0$ . Then, assuming that  $S(k)$  is true for all  $k$  satisfying

$$n_0 \leq k < n, \tag{2.5.1}$$

we prove that  $S(n)$  is true. In inequality (2.5.1),  $k$  indexes a statement  $S(k)$  that is an *arbitrary* predecessor of the statement  $S(n)$  (thus  $k < n$ ), which we are to prove true. In inequality (2.5.1),  $n_0 \leq k$  ensures that  $k$  is in the domain of discourse

$$\{n_0, n_0 + 1, n_0 + 2, \dots\}.$$

The two forms of mathematical induction are logically equivalent (see Exercise 38).

We present several examples that illustrate the use of the Strong Form of Mathematical Induction.

### Example 2.5.1

Use mathematical induction to show that postage of 4 cents or more can be achieved by using only 2-cent and 5-cent stamps.

**SOLUTION Discussion** Consider the Inductive Step, where we want to prove that  $n$ -cents postage can be achieved using only 2-cent and 5-cent stamps. It would be particularly easy to prove this statement if we could assume that we can make postage of  $n - 2$  cents. We could then simply add a 2-cent stamp to make  $n$ -cents postage. How simple! If we use the Strong Form of Mathematical Induction, we *can* assume the truth of the statement for all  $k < n$ . In particular, we can assume the truth of the statement for  $k = n - 2$ . Thus the Strong Form of Mathematical Induction allows us to give a correct proof based on our informal reasoning.

In this example,  $n_0$  in inequality (2.5.1) is equal to 4. When we take  $k = n - 2$ , to ensure that  $n_0 \leq k$ , that is  $4 \leq n - 2$ , we must have  $6 \leq n$ . Now  $n = 4$  is the Basis Step. What about  $n = 5$ ? We explicitly prove this case. By convention, we add the case  $n = 5$  to the Basis Step; thus the cases  $n = 4$  and  $n = 5$  become the Basis Steps. In general, if the Inductive Step assumes that the case  $n - p$  is true (in this example,  $p = 2$ ), there will be  $p$  Basis Steps:  $n = n_0, n = n_0 + 1, \dots, n = n_0 + p - 1$ .

#### Basis Steps ( $n = 4, n = 5$ )

We can make 4-cents postage by using two 2-cent stamps. We can make 5-cents postage by using one 5-cent stamp. The Basis Steps are verified.

#### Inductive Step

We assume that  $n \geq 6$  and that postage of  $k$  cents or more can be achieved by using only 2-cent and 5-cent stamps for  $4 \leq k < n$ .

By the inductive assumption, we can make postage of  $n - 2$  cents. We add a 2-cent stamp to make  $n$ -cents postage. The Inductive Step is complete. 

### Example 2.5.2

When an element of a sequence is defined in terms of some of its predecessors, the Strong Form of Mathematical Induction is sometimes useful to prove a property of the sequence. For example, suppose that the sequence  $c_1, c_2, \dots$  is defined by the equations<sup>†</sup>

$$c_1 = 0, \quad c_n = c_{\lfloor n/2 \rfloor} + n \text{ for all } n > 1.$$

As examples,

$$\begin{aligned} c_2 &= c_{\lfloor 2/2 \rfloor} + 2 = c_{\lfloor 1 \rfloor} + 2 = c_1 + 2 = 0 + 2 = 2, \\ c_3 &= c_{\lfloor 3/2 \rfloor} + 3 = c_{\lfloor 1.5 \rfloor} + 3 = c_1 + 3 = 0 + 3 = 3, \\ c_4 &= c_{\lfloor 4/2 \rfloor} + 4 = c_{\lfloor 2 \rfloor} + 4 = c_2 + 4 = 2 + 4 = 6, \\ c_5 &= c_{\lfloor 5/2 \rfloor} + 5 = c_{\lfloor 2.5 \rfloor} + 5 = c_2 + 5 = 2 + 5 = 7. \end{aligned}$$

Use strong induction to prove that  $c_n < 2n$ , for all  $n \geq 1$ .

**SOLUTION Discussion** In this example,  $n_0$  in inequality (2.5.1) is equal to 1 and (2.5.1) becomes  $1 \leq k < n$ . In particular, since  $c_n$  is defined in terms of  $c_{\lfloor n/2 \rfloor}$ , in the Inductive Step we assume the truth of the statement for  $k = \lfloor n/2 \rfloor$ . Inequality (2.5.1)

<sup>†</sup>The *floor of  $x$* ,  $\lfloor x \rfloor$ , is the greatest integer less than or equal to  $x$  (see Section 3.1). Informally, we are “rounding down.” Examples:  $\lfloor 2.3 \rfloor = 2$ ,  $\lfloor 5 \rfloor = 5$ ,  $\lfloor -2.7 \rfloor = -3$ .

then becomes  $1 \leq \lfloor n/2 \rfloor < n$ . Because  $n/2 < n$ , it follows that  $\lfloor n/2 \rfloor < n$ . If  $n \geq 2$ , then  $1 \leq n/2$  and so  $1 \leq \lfloor n/2 \rfloor$ . Therefore if  $n \geq 2$  and  $k = \lfloor n/2 \rfloor$ , inequality (2.5.1) is satisfied. Thus the Basis Step is  $n = 1$ .

### Basis Step ( $n = 1$ )

Since  $c_1 = 0 < 2 = 2 \cdot 1$ , the Basis Step is verified.

### Inductive Step

We assume that  $c_k < 2k$ , for all  $k$ ,  $1 \leq k < n$ , and prove that  $c_n < 2n$ ,  $n > 1$ . Since  $1 < n$ ,  $2 \leq n$ . Thus  $1 \leq n/2 < n$ . Therefore  $1 \leq \lfloor n/2 \rfloor < n$  and taking  $k = \lfloor n/2 \rfloor$ , we see that  $k$  satisfies inequality (2.5.1). By the inductive assumption

$$c_{\lfloor n/2 \rfloor} = c_k < 2k = 2\lfloor n/2 \rfloor.$$

Now

$$c_n = c_{\lfloor n/2 \rfloor} + n < 2\lfloor n/2 \rfloor + n \leq 2(n/2) + n = 2n.$$

The Inductive Step is complete. 

### Example 2.5.3

Define the sequence  $c_1, c_2, \dots$  by the equations

$$c_1 = 1, \quad c_n = c_{\lfloor n/2 \rfloor} + n^2 \text{ for all } n > 1.$$

Suppose that we want to prove a statement for all  $n \geq 2$  involving  $c_n$ . The Inductive Step will assume the truth of the statement involving  $c_{\lfloor n/2 \rfloor}$ . What are the Basis Steps?

**SOLUTION** In this example,  $n_0$  in inequality (2.5.1) is equal to 2 and (2.5.1) becomes  $2 \leq k < n$ . In the Inductive Step, we assume the truth of the statement for  $k = \lfloor n/2 \rfloor$ . Inequality (2.5.1) then becomes  $2 \leq \lfloor n/2 \rfloor < n$ . Because  $n/2 < n$ , it follows that  $\lfloor n/2 \rfloor < n$ .

If  $n = 3$ , then  $2 > \lfloor n/2 \rfloor$ . Thus we must add  $n = 3$  to the Basis Step ( $n = 2$ ). If  $n \geq 4$ , then  $2 \leq n/2$  and so  $2 \leq \lfloor n/2 \rfloor$ . Therefore if  $n \geq 4$  and  $k = \lfloor n/2 \rfloor$ , inequality (2.5.1) is satisfied. Thus the Basis Steps are  $n = 2$  and  $n = 3$ . 

### Example 2.5.4

Suppose that we insert parentheses and then multiply the  $n$  numbers  $a_1 a_2 \cdots a_n$ . For example, if  $n = 4$ , we might insert the parentheses as shown:

$$(a_1 a_2)(a_3 a_4). \tag{2.5.2}$$

Here we would first multiply  $a_1$  by  $a_2$  to obtain  $a_1 a_2$  and  $a_3$  by  $a_4$  to obtain  $a_3 a_4$ . We would then multiply  $a_1 a_2$  by  $a_3 a_4$  to obtain  $(a_1 a_2)(a_3 a_4)$ . Notice that the number of multiplications is three. Use strong induction to prove that if we insert parentheses in any manner whatsoever and then multiply the  $n$  numbers  $a_1 a_2 \cdots a_n$ , we perform  $n - 1$  multiplications.

### SOLUTION

#### Basis Step ( $n = 1$ )

We need 0 multiplications to compute  $a_1$ . The Basis Step is verified.

#### Inductive Step

We assume that for all  $k$ ,  $1 \leq k < n$ , it takes  $k - 1$  multiplications to compute the product of  $k$  numbers if parentheses are inserted in any manner whatsoever. We must prove that it takes  $n$  multiplications to compute the product  $a_1 a_2 \cdots a_n$  if parentheses are inserted in any manner whatsoever.

Suppose that parentheses are inserted in the product  $a_1 a_2 \cdots a_n$ . Consider the final multiplication, which looks like  $(a_1 \cdots a_t)(a_{t+1} \cdots a_n)$ , for some  $t$ ,  $1 \leq t < n$ . [For example, in equation (2.5.2),  $t = 2$ .] There are  $t$  terms in the first set of parentheses,  $1 \leq t < n$ , and  $n - t$  terms in the second set of parentheses,  $1 \leq n - t < n$ . By the inductive assumption, it takes  $t - 1$  multiplications to compute  $a_1 \cdots a_t$  and  $n - t - 1$  multiplications to compute  $a_{t+1} \cdots a_n$ , regardless of how the parentheses are inserted. It takes one additional multiplication to multiply  $a_1 \cdots a_t$  by  $a_{t+1} \cdots a_n$ . Thus the total number of multiplications is

$$(t - 1) + (n - t - 1) + 1 = n - 1.$$

The Inductive Step is complete. 

### Well-Ordering Property

The **Well-Ordering Property for nonnegative integers** states that every nonempty set of nonnegative integers has a least element. This property is equivalent to the two forms of induction (see Exercises 36–38). We use the Well-Ordering Property to prove something familiar from long division: When we divide an integer  $n$  by a positive integer  $d$ , we obtain a quotient  $q$  and a remainder  $r$  satisfying  $0 \leq r < d$  so that  $n = dq + r$ .

#### Example 2.5.5

When we divide  $n = 74$  by  $d = 13$

$$\begin{array}{r} 5 \\ 13) \overline{74} \\ 65 \\ \hline 9 \end{array}$$

we obtain the quotient  $q = 5$  and the remainder  $r = 9$ . Notice that  $r$  satisfies  $0 \leq r < d$ ; that is,  $0 \leq 9 < 13$ . We have

$$n = 74 = 13 \cdot 5 + 9 = dq + r. \quad \blacktriangleleft$$

#### Theorem 2.5.6

#### Quotient-Remainder Theorem

If  $d$  and  $n$  are integers,  $d > 0$ , there exist integers  $q$  (quotient) and  $r$  (remainder) satisfying

$$n = dq + r \quad 0 \leq r < d.$$

Furthermore,  $q$  and  $r$  are unique; that is, if

$$n = dq_1 + r_1 \quad 0 \leq r_1 < d$$

and

$$n = dq_2 + r_2 \quad 0 \leq r_2 < d,$$

then  $q_1 = q_2$  and  $r_1 = r_2$ .

**Discussion** We can devise a proof of Theorem 2.5.6 by looking carefully at the technique used in long division. Why is 5 the quotient in Example 2.5.5? Because  $q = 5$  makes the remainder  $n - dq$  nonnegative and as small as possible. If, for example,  $q = 3$ , the remainder would be  $n - dq = 74 - 13 \cdot 3 = 35$ , which is too large. As another example, if  $q = 6$ , the remainder would be  $n - dq = 74 - 13 \cdot 6 = -4$ , which is negative. The existence of a smallest, nonnegative remainder  $n - dq$  is guaranteed by the Well-Ordering Property.

**Proof** Let

$$X = \{n - dk \mid n - dk \geq 0, k \in \mathbf{Z}\}.$$

We show that  $X$  is nonempty using proof by cases. If  $n \geq 0$ , then  $n - d \cdot 0 = n \geq 0$  so  $n$  is in  $X$ . Suppose that  $n < 0$ . Since  $d$  is a positive integer,  $1 - d \leq 0$ . Thus  $n - dn = n(1 - d) \geq 0$ . In this case,  $n - dn$  is in  $X$ . Therefore  $X$  is nonempty.

Since  $X$  is a nonempty set of nonnegative integers, by the Well-Ordering Property,  $X$  has a smallest element, which we denote  $r$ . We let  $q$  denote the specific value of  $k$  for which  $r = n - dq$ . Then  $n = dq + r$ .

Since  $r$  is in  $X$ ,  $r \geq 0$ . We use proof by contradiction to show that  $r < d$ . Suppose that  $r \geq d$ . Then

$$n - d(q + 1) = n - dq - d = r - d \geq 0.$$

Thus  $n - d(q + 1)$  is in  $X$ . Also,  $n - d(q + 1) = r - d < r$ . But  $r$  is the smallest integer in  $X$ . This contradiction shows that  $r < d$ .

We have shown that if  $d$  and  $n$  are integers,  $d > 0$ , there exist integers  $q$  and  $r$  satisfying

$$n = dq + r \quad 0 \leq r < d.$$

We turn now to the uniqueness of  $q$  and  $r$ . Suppose that

$$n = dq_1 + r_1 \quad 0 \leq r_1 < d$$

and

$$n = dq_2 + r_2 \quad 0 \leq r_2 < d.$$

We must show that  $q_1 = q_2$  and  $r_1 = r_2$ . Subtracting the previous equations, we obtain

$$0 = n - n = (dq_1 + r_1) - (dq_2 + r_2) = d(q_1 - q_2) - (r_2 - r_1),$$

which can be rewritten

$$d(q_1 - q_2) = r_2 - r_1.$$

The preceding equation shows that  $d$  divides  $r_2 - r_1$ . However, because  $0 \leq r_1 < d$  and  $0 \leq r_2 < d$ ,

$$-d < r_2 - r_1 < d.$$

But the only integer strictly between  $-d$  and  $d$  divisible by  $d$  is 0. Therefore,  $r_1 = r_2$ . Thus,  $d(q_1 - q_2) = 0$ ; hence,  $q_1 = q_2$ . The proof is complete. ◀

Notice that in Theorem 2.5.6 the remainder  $r$  is zero if and only if  $d$  divides  $n$ .

## 2.5 Problem-Solving Tips

In the Inductive Step of the Strong Form of Mathematical Induction, your goal is to prove case  $n$ . To do so, you can assume *all* preceding cases (not just the immediately preceding case as in Section 2.4). You could always use the Strong Form of Mathematical Induction. If it happens that you needed only the immediately preceding case in the Inductive Step, you merely used the form of mathematical induction of Section 2.4. However, assuming all previous cases potentially gives you more to work with in proving case  $n$ .

In the Inductive Step of the Strong Form of Mathematical Induction, when you assume that the statement  $S(k)$  is true, you must be sure that  $k$  is in the domain of discourse of the propositional function  $S(n)$ . In the terminology of this section, you must be sure that  $n_0 \leq k$  (see Examples 2.5.1 and 2.5.2).

In the Inductive Step of the Strong Form of Mathematical Induction, if you assume that case  $n-p$  is true, there will be  $p$  Basis Steps:  $n = n_0, n = n_0 + 1, \dots, n = n_0 + p - 1$ .

In general, the key to devising a proof using the Strong Form of Mathematical Induction is to find smaller cases “within” case  $n$ . For example, the smaller cases in Example 2.5.4 are the parenthesized products  $(a_1 \cdots a_t)$  and  $(a_{t+1} \cdots a_n)$  for  $1 \leq t < n$ .

## 2.5 Review Exercises

1. State the Strong Form of Mathematical Induction.
2. State the Well-Ordering Property.
3. State the Quotient-Remainder Theorem.

## 2.5 Exercises

1. Show that postage of 6 cents or more can be achieved by using only 2-cent and 7-cent stamps.
2. Show that postage of 24 cents or more can be achieved by using only 5-cent and 7-cent stamps.
3. Show that postage of 12 cents or more can be achieved by using only 3-cent and 7-cent stamps.
4. Use the

If  $S(n)$  is true, then  $S(n + 1)$  is true

form of the Inductive Step to prove the statement in Example 2.5.1.

5. Use the

If  $S(n)$  is true, then  $S(n + 1)$  is true

form of the Inductive Step to prove the statement in Exercise 1.

6. Use the

If  $S(n)$  is true, then  $S(n + 1)$  is true

form of the Inductive Step to prove the statement in Exercise 2.

*Exercises 7 and 8 refer to the sequence  $c_1, c_2, \dots$  defined by the equations*

$$c_1 = 0, \quad c_n = c_{\lfloor n/2 \rfloor} + n^2 \text{ for all } n > 1.$$

7. Suppose that we want to prove a statement for all  $n \geq 3$  involving  $c_n$ . The Inductive Step will assume the truth of the statement involving  $c_{\lfloor n/2 \rfloor}$ . What are the Basis Steps?
8. Suppose that we want to prove a statement for all  $n \geq 4$  involving  $c_n$ . The Inductive Step will assume the truth of the statement involving  $c_{\lfloor n/2 \rfloor}$ . What are the Basis Steps?
9. Define the sequence  $c_1, c_2, \dots$  by the equations

$$c_1 = c_2 = 0, \quad c_n = c_{\lfloor n/3 \rfloor} + n \text{ for all } n > 2.$$

Suppose that we want to prove a statement for all  $n \geq 2$  involving  $c_n$ . The Inductive Step will assume the truth of the statement involving  $c_{\lfloor n/3 \rfloor}$ . What are the Basis Steps?

*Exercises 10 and 11 refer to the sequence  $c_1, c_2, \dots$  defined by the equations*

$$c_1 = 0, \quad c_n = c_{\lfloor n/2 \rfloor} + n^2 \text{ for all } n > 1.$$

10. Compute  $c_2, c_3, c_4$ , and  $c_5$ .
11. Prove that  $c_n < 4n^2$  for all  $n \geq 1$ .

*Exercises 12–14 refer to the sequence  $c_1, c_2, \dots$  defined by the equations*

$$c_1 = 0, \quad c_n = 4c_{\lfloor n/2 \rfloor} + n \text{ for all } n > 1.$$

12. Compute  $c_2, c_3, c_4$ , and  $c_5$ .
13. Prove that  $c_n \leq 4(n-1)^2$  for all  $n \geq 1$ .
14. Prove that  $(n+1)^2/8 < c_n$  for all  $n \geq 2$ . Hint:  $\lfloor n/2 \rfloor \geq (n-1)/2$  for all  $n$ .
15. Define the sequence  $c_0, c_1, \dots$  by the equations

$$c_0 = 0, \quad c_n = c_{\lfloor n/2 \rfloor} + 3 \text{ for all } n > 0.$$

What is wrong with the following “proof” that  $c_n \leq 2n$  for all  $n \geq 3$ ? (You should verify that it is *false* that  $c_n \leq 2n$  for all  $n \geq 3$ .)

We use the Strong Form of Mathematical Induction.

### Basis Step ( $n = 3$ )

We have

$$c_3 = c_1 + 3 = (c_0 + 3) + 3 = 6 \leq 2 \cdot 3.$$

The Basis Step is verified.

### Inductive Step

Assume that  $c_k \leq 2k$  for all  $k < n$ . Then

$$c_n = c_{\lfloor n/2 \rfloor} + 3 \leq 2\lfloor n/2 \rfloor + 3 \leq 2(n/2) + 3 = n + 3 < n + n = 2n.$$

(Since  $3 < n, n + 3 < n + n$ .) The Inductive Step is complete.

16. Suppose that we have two piles of cards each containing  $n$  cards. Two players play a game as follows. Each player, in turn, chooses one pile and then removes any number of cards, but at least one, from the chosen pile. The player who removes the last card wins the game. Show that the second player can always win the game.

In Exercises 17–22, find the quotient  $q$  and remainder  $r$  as in Theorem 2.5.6 when  $n$  is divided by  $d$ .

17.  $n = 47, d = 9$       18.  $n = -47, d = 9$   
 19.  $n = 7, d = 9$       20.  $n = -7, d = 9$   
 21.  $n = 0, d = 9$       22.  $n = 47, d = 47$

The Egyptians of antiquity expressed a fraction as a sum of fractions whose numerators were 1. For example,  $5/6$  might be expressed as

$$\frac{5}{6} = \frac{1}{2} + \frac{1}{3}.$$

We say that a fraction  $p/q$ , where  $p$  and  $q$  are positive integers, is in Egyptian form if

$$\frac{p}{q} = \frac{1}{n_1} + \frac{1}{n_2} + \cdots + \frac{1}{n_k}, \quad (2.5.3)$$

where  $n_1, n_2, \dots, n_k$  are positive integers satisfying  $n_1 < n_2 < \cdots < n_k$ .

23. Show that the representation (2.5.3) need not be unique by representing  $5/6$  in two different ways.  
 24. Show that the representation (2.5.3) is never unique.  
 25. By completing the following steps, give a proof by induction on  $p$  to show that every fraction  $p/q$  with  $0 < p/q < 1$  may be expressed in Egyptian form.  
 (a) Verify the Basis Step ( $p = 1$ ).  
 (b) Suppose that  $0 < p/q < 1$  and that all fractions  $i/q'$ , with  $1 \leq i < p$  and  $q'$  arbitrary, can be expressed in Egyptian form. Choose the smallest positive integer  $n$  with  $1/n \leq p/q$ . Show that

$$n > 1 \quad \text{and} \quad \frac{p}{q} < \frac{1}{n-1}.$$

- (c) Show that if  $p/q = 1/n$ , the proof is complete.  
 (d) Assume that  $1/n < p/q$ . Let

$$p_1 = np - q \quad \text{and} \quad q_1 = nq.$$

Show that

$$\frac{p_1}{q_1} = \frac{p}{q} - \frac{1}{n}, \quad 0 < \frac{p_1}{q_1} < 1, \quad \text{and} \quad p_1 < p.$$

Conclude that

$$\frac{p_1}{q_1} = \frac{1}{n_1} + \frac{1}{n_2} + \cdots + \frac{1}{n_k}$$

with  $n_1, n_2, \dots, n_k$  distinct.

- (e) Show that  $p_1/q_1 < 1/n$ .

- (f) Show that

$$\frac{p}{q} = \frac{1}{n} + \frac{1}{n_1} + \cdots + \frac{1}{n_k}$$

and  $n, n_1, \dots, n_k$  are distinct.

26. Use the method of the preceding exercise to find Egyptian forms of  $3/8, 5/7$ , and  $13/19$ .  
 \*27. Show that any fraction  $p/q$ , where  $p$  and  $q$  are positive integers, can be written in Egyptian form. (We are not assuming that  $p/q < 1$ ).  
 \*28. Show that any  $n \times n$  deficient board can be tiled with trominoes if  $n$  is odd,  $n > 5$ , and 3 divides  $n^2 - 1$ . Hint: Use the ideas suggested in the hint for Exercise 38, Section 2.4.  
 \*29. Show that any  $n \times n$  deficient board can be tiled with trominoes if  $n$  is even,  $n > 8$ , and 3 divides  $n^2 - 1$ . Hint: Use the fact that a  $4 \times 4$  deficient board can be tiled with trominoes, Exercise 28, and Exercise 36, Section 2.4.  
 \*30. Show that any  $m \times n$  deficient rectangle,  $2 \leq m \leq n$ , can be tiled with trominoes if 3 divides  $mn - 1$ , neither side has length 2 unless both of them do, and  $m \neq 5$ .  
 31. Give an example of an  $m \times n$  rectangle with two squares missing, where 3 divides  $mn - 2$ , that can be tiled with trominoes.  
 32. Give an example of an  $m \times n$  rectangle with two squares missing, where 3 divides  $mn - 2$ , that cannot be tiled with trominoes.  
 \*33. Which  $m \times n$  rectangles with two squares missing, where 3 divides  $mn - 2$ , can be tiled with trominoes?  
 34. Give an alternative proof of the existence of  $q$  and  $r$  in Theorem 2.5.6 for the case  $n \geq 0$  by first showing that the set  $X$  consisting of all integers  $k$  where  $dk > n$  is a nonempty set of nonnegative integers, then showing that  $X$  has a least element, and finally analyzing the least element of  $X$ .  
 35. Give an alternative proof of the existence of  $q$  and  $r$  in Theorem 2.5.6 using the form of mathematical induction where the Inductive Step is “if  $S(n)$  is true, then  $S(n+1)$  is true.” Hint: First assume that  $n > 0$ . Treat the case  $n = 0$  separately. Reduce the case  $n < 0$  to the case  $n > 0$ .  
 \*36. Assume the form of mathematical induction where the Inductive Step is “if  $S(n)$  is true, then  $S(n+1)$  is true.” Prove the Well-Ordering Property.  
 \*37. Assume the Well-Ordering Property. Prove the Strong Form of Mathematical Induction.  
 \*38. Show that the Strong Form of Mathematical Induction and the form of mathematical induction where the Inductive Step is “if  $S(n)$  is true, then  $S(n+1)$  is true” are equivalent. That is, assume the Strong Form of Mathematical Induction and prove the alternative form; then assume the alternative form and prove the Strong Form of Mathematical Induction.

## Chapter 2 Notes

[D'Angelo; Solow] address the problem of how to construct proofs. Tiling with polyominoes is the subject of the book by [Martin].

The “Fallacies, Flaws, and Flimflam” section of *The College Mathematics Journal*, published by the Mathematical Association of America, contains examples of mathematical mistakes, fallacious proofs, and faulty reasoning.

## Chapter 2 Review

### Section 2.1

1. Mathematical system
2. Axiom
3. Definition
4. Undefined term
5. Theorem
6. Proof
7. Lemma
8. Direct proof
9. Even integer
10. Odd integer
11. Subproof
12. Disproving a universally quantified statement
13. Begging the question
14. Circular reasoning

### Section 2.2

15. Proof by contradiction
16. Indirect proof
17. Proof by contrapositive
18. Proof by cases
19. Exhaustive proof
20. Proving an if-and-only-if statement
21. Proving several statements are equivalent
22. Existence proof
23. Constructive existence proof
24. Nonconstructive existence proof

### Section 2.3

25. Resolution proof; uses: if  $p \vee q$  and  $\neg p \vee r$  are both true, then  $q \vee r$  is true.

26. Clause: consists of terms separated by *or*'s, where each term is a variable or a negation of a variable.

### Section 2.4

27. Principle of Mathematical Induction
28. Basis Step: prove true for the first instance.
29. Inductive Step: assume true for instance  $n$ ; then prove true for instance  $n + 1$ .
30.  $n$  factorial:  $n! = n(n - 1) \cdots 1, 0! = 1$
31. Formula for the sum of the first  $n$  positive integers:  

$$1 + 2 + \cdots + n = \frac{n(n + 1)}{2}$$
32. Formula for the geometric sum:  

$$ar^0 + ar^1 + \cdots + ar^n = \frac{a(r^{n+1} - 1)}{r - 1}, \quad r \neq 1$$

### Section 2.5

33. Strong Form of Mathematical Induction
34. Basis Step for the Strong Form of Mathematical Induction: prove true for the first instance.
35. Inductive Step for the Strong Form of Mathematical Induction: assume true for all instances less than  $n$ ; then prove true for instance  $n$ .
36. Well-Ordering Property: every nonempty set of nonnegative integers has a least element.
37. Quotient-Remainder Theorem: If  $d$  and  $n$  are integers,  $d > 0$ , there exist integers  $q$  (quotient) and  $r$  (remainder) satisfying  $n = dq + r$ ,  $0 \leq r < d$ . Furthermore,  $q$  and  $r$  are unique.

## Chapter 2 Self-Test

1. Distinguish between the terms *axiom* and *definition*.
2. What is the difference between a direct proof and a proof by contradiction?
3. Show, by giving a proof by contradiction, that if four teams play seven games, some pair of teams plays at least two times.

4. Prove that for all rational numbers  $x$  and  $y$ ,  $y \neq 0$ ,  $x/y$  is rational.
5. Use proof by cases to prove that

$$\min\{\min\{a, b\}, c\} = \min\{a, \min\{b, c\}\}$$

for all real numbers  $a$ ,  $b$ , and  $c$ .

Use mathematical induction to prove that the statements in Exercises 6–9 are true for every positive integer  $n$ .

6.  $2 + 4 + \cdots + 2n = n(n + 1)$
7.  $2^2 + 4^2 + \cdots + (2n)^2 = \frac{2n(n + 1)(2n + 1)}{3}$
8.  $\frac{1}{2!} + \frac{2}{3!} + \cdots + \frac{n}{(n + 1)!} = 1 - \frac{1}{(n + 1)!}$
9.  $2^{n+1} < 1 + (n + 1)2^n$

10. Prove that for all integers  $m$  and  $n$ , if  $m$  and  $m - n$  are odd, then  $n$  is even.
11. Prove that the following are equivalent for sets  $A$  and  $B$ :

$$(a) A \subseteq B \quad (b) A \cap \overline{B} = \emptyset \quad (c) A \cup B = B$$

12. Prove that for all sets  $X$ ,  $Y$ , and  $Z$ , if  $X \subseteq Y$  and  $Y \subset Z$ , then  $X \subset Z$ .
13. Find the quotient  $q$  and remainder  $r$  as in Theorem 2.5.6 when  $n = 101$  is divided by  $d = 11$ .

Exercises 14 and 15 refer to the sequence  $c_1, c_2, \dots$  defined by the equations

$$c_1 = 0, \quad c_n = 2c_{\lfloor n/2 \rfloor} + n \text{ for all } n > 1.$$

14. Compute  $c_2, c_3, c_4$ , and  $c_5$ .
15. Prove that  $c_n \leq n \lg n$  for all  $n \geq 1$ .
16. Use the Well-Ordering Property to show that any nonempty set  $X$  of nonnegative integers that has an upper bound contains a largest element. Hint: Consider the set of integer upper bounds for  $X$ .
17. Find an expression, which is the *and* of clauses, equivalent to  $(p \vee q) \rightarrow r$ .
18. Find an expression, which is the *and* of clauses, equivalent to  $(p \vee \neg q) \rightarrow \neg rs$ .
19. Use resolution to prove

$$\begin{array}{c} \neg p \vee q \\ \neg q \vee \neg r \\ \hline p \vee \neg r \\ \hline \therefore \neg r \end{array}$$

20. Reprove Exercise 19 using resolution and proof by contradiction.

## Chapter 2 Computer Exercises

1. Implement proof by resolution as a program.
2. Write a program that gives an Egyptian form of a fraction.



## Chapter 3

# FUNCTIONS, SEQUENCES, AND RELATIONS

- 
- 3.1** Functions
  - 3.2** Sequences and Strings
  - 3.3** Relations
  - 3.4** Equivalence Relations
  - 3.5** Matrices of Relations
  - †3.6** Relational Databases

All of mathematics, as well as subjects that rely on mathematics, such as computer science and engineering, make use of functions, sequences, and relations.

A function assigns to each member of a set  $X$  exactly one member of a set  $Y$ . Functions are used extensively in discrete mathematics; for example, functions are used to analyze the time needed to execute algorithms.

A sequence is a special kind of function. A list of the letters as they appear in a word is an example of a sequence. Unlike a set, a sequence takes order into account. (Order is obviously important since, for example, *form* and *from* are different words.)

Relations generalize the notion of functions. A relation is a set of ordered pairs. The presence of the ordered pair  $(a, b)$  in a relation is interpreted as indicating a relationship from  $a$  to  $b$ . The relational database model that helps users access information in a database (a collection of records manipulated by a computer) is based on the concept of relation.

## 3.1 Functions

Credit card numbers typically consist of 13, 15, or 16 digits. For example,

$$4690\ 3582\ 1375\ 4657 \quad (3.1.1)$$

is a hypothetical credit card number. The first digit designates the system. In (3.1.1), the first digit, 4, shows that the card would be a Visa card. The following digits specify other information such as the account number and the bank number. (The precise meaning depends on the type of card.) The last digit is special; it is computed from the preceding digits and is called a *check digit*. In (3.1.1), the check digit is 7 and is computed from the preceding digits 4690 3582 1375 465. Credit card check digits are used to identify certain erroneous card numbers. It is not a security measure, but rather it is used to help detect errors such as giving a credit card number over the phone and having it transcribed improperly or detecting an error in entering a credit card number while ordering a product online.

---

<sup>†</sup>This section can be omitted without loss of continuity.

The check digit is computed as follows. Starting from the right and skipping the check digit, double every other number. If the result of doubling is a two-digit number, add the digits; otherwise, use the original digit. The other digits are not modified.

|                                   |                                  |
|-----------------------------------|----------------------------------|
| 4 6 9 0 3 5 8 2 1 3 7 5 4 6 5     |                                  |
| ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓     | Double every other digit.        |
| 8 6 18 0 6 5 16 2 2 3 14 5 8 6 10 |                                  |
| ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓     | Add digits of two-digit numbers. |
| 8 6 9 0 6 5 7 2 2 3 5 5 8 6 1     |                                  |

Sum the resulting digits

$$8 + 6 + 9 + 0 + 6 + 5 + 7 + 2 + 2 + 3 + 5 + 5 + 8 + 6 + 1 = 73.$$

If the last digit of the sum is 0, the check digit is 0. Otherwise, subtract the last digit of the sum from 10 to get the check digit,  $10 - 3 = 7$ . Verify the check digit on your favorite Visa, MasterCard, American Express, or Diners Club card. This method of calculating a check digit is called the *Luhn algorithm*. It is named after Hans Peter Luhn (1896–1964), who invented it while at IBM. Although originally patented, it is now in the public domain and is widely used.

One common error in copying a number is to change one digit. Each undoubled digit contributes a unique value to the sum ( $0 \rightarrow 0$ ,  $1 \rightarrow 1$ , etc.). Each doubled digit also contributes a unique value to the sum ( $0 \rightarrow 0$ ,  $1 \rightarrow 2, \dots, 4 \rightarrow 8$ ,  $5 \rightarrow 1$ ,  $6 \rightarrow 3, \dots, 9 \rightarrow 9$ ). Thus if a single digit is changed in a credit card number, the sum used in the Luhn algorithm will change by an absolute amount less than 10, and the check digit will change. In the preceding example if 1 is changed to 7, the Luhn algorithm calculation becomes

|                                    |                                  |
|------------------------------------|----------------------------------|
| 4 6 9 0 3 5 8 2 7 3 7 5 4 6 5      |                                  |
| ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓      | Double every other digit         |
| 8 6 18 0 6 5 16 2 14 3 14 5 8 6 10 |                                  |
| ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓      | Add digits of two-digit numbers. |
| 8 6 9 0 6 5 7 2 5 3 5 5 8 6 1      |                                  |

and the sum becomes

$$8 + 6 + 9 + 0 + 6 + 5 + 7 + 2 + 5 + 3 + 5 + 5 + 8 + 6 + 1 = 76.$$

Therefore the check digit changes to 4. Thus, if 1 is inadvertently transcribed as 7, the error will be detected.

Another common error is transposition of adjacent digits. For example, if 82 is inadvertently written as 28, the error will be detected by the Luhn algorithm because the check digit will change (check this). In fact, the Luhn algorithm will detect every transposition of adjacent digits except for 90 and 09 (see Computer Exercise 4).

The Luhn algorithm gives an example of a **function**. A function assigns to each member of a set  $X$  exactly one member of a set  $Y$ . (The sets  $X$  and  $Y$  may or may not be the same.) The Luhn algorithm assigns to each integer 10 or greater (so there is a number available to compute a check digit) a single-digit integer, the check digit. In the preceding example, the integer 469035821375465 is assigned the value 7, and the integer 469035827375465 is assigned the value 4. We can represent these assignments as ordered pairs:

$$(469035821375465, 7) \text{ and } (469035827375465, 4).$$

### Go Online

For more on functions, see [goo.gl/V3y4ps](http://goo.gl/V3y4ps)

Formally, we *define* a function to be a particular kind of set of ordered pairs.

**Definition 3.1.1** ► Let  $X$  and  $Y$  be sets. A *function*  $f$  from  $X$  to  $Y$  is a subset of the Cartesian product  $X \times Y$  having the property that for each  $x \in X$ , there is exactly one  $y \in Y$  with  $(x, y) \in f$ . We sometimes denote a function  $f$  from  $X$  to  $Y$  as  $f: X \rightarrow Y$ .

The set  $X$  is called the *domain of  $f$*  and the set  $Y$  is called the *codomain of  $f$* . The set

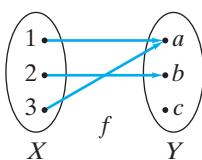
$$\{y \mid (x, y) \in f\}$$

(which is a subset of the codomain  $Y$ ) is called the *range of  $f$* . ▲

### Example 3.1.2

For the check digit function, the domain is the set of positive integers 10 or greater and the range is the set of single-digit integers. We can take the codomain to be any set containing the set of single-digit integers, for example, the set of nonnegative integers. ▲

### Example 3.1.3



**Figure 3.1.1** The arrow diagram of the function of Example 3.1.3. There is exactly one arrow from each element in  $X$ .

The set  $f = \{(1, a), (2, b), (3, a)\}$  is a function from  $X = \{1, 2, 3\}$  to  $Y = \{a, b, c\}$ . Each element of  $X$  is assigned a unique value in  $Y$ : 1 is assigned the unique value  $a$ ; 2 is assigned the unique value  $b$ ; and 3 is assigned the unique value  $a$ . We can depict the situation as shown in Figure 3.1.1, where an arrow from  $j$  to  $x$  means that we assign the letter  $x$  to the integer  $j$ . We call a picture such as Figure 3.1.1 an **arrow diagram**. For an arrow diagram to be a function, Definition 3.1.1 requires that there is exactly one arrow from each element in the domain. Notice that Figure 3.1.1 has this property.

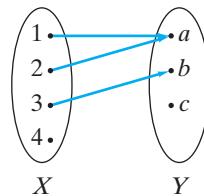
Definition 3.1.1 allows us to reuse elements in  $Y$ . For the function  $f$ , the element  $a$  in  $Y$  is used twice. Further, Definition 3.1.1 does *not* require us to use all the elements in  $Y$ . No element in  $X$  is assigned to the element  $c$  in  $Y$ . The domain of  $f$  is  $X$ , the codomain of  $f$  is  $Y$ , and the range of  $f$  is  $\{a, b\}$ . ▲

### Example 3.1.4

The set

$$\{(1, a), (2, a), (3, b)\} \quad (3.1.2)$$

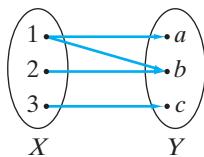
is not a function from  $X = \{1, 2, 3, 4\}$  to  $Y = \{a, b, c\}$  because the element 4 in  $X$  is not assigned to an element in  $Y$ . It is also apparent from the arrow diagram (see Figure 3.1.2) that this set is not a function because there is no arrow from 4. The set (3.1.2) *is* a function from  $X' = \{1, 2, 3\}$  to  $Y = \{a, b, c\}$ .



**Figure 3.1.2** The arrow diagram of the set in Example 3.1.4, which is not a function because there is no arrow from 4. ▲

### Example 3.1.5

The set  $\{(1, a), (2, b), (3, c), (1, b)\}$  is not a function from  $X = \{1, 2, 3\}$  to  $Y = \{a, b, c\}$  because 1 is not assigned a *unique* element in  $Y$  (1 is assigned the values  $a$  and  $b$ ). It is also apparent from the arrow diagram (see Figure 3.1.3) that this set is not a function because there are two arrows from 1.



**Figure 3.1.3** The arrow diagram of the set in Example 3.1.5, which is not a function because there are two arrows from 1.

Given a function  $f$  from  $X$  to  $Y$ , according to Definition 3.1.1, for each element  $x$  in the domain  $X$ , there is exactly one  $y$  in the codomain  $Y$  with  $(x, y) \in f$ . This unique value  $y$  is denoted  $f(x)$ . In other words,  $f(x) = y$  is another way to write  $(x, y) \in f$ .

### Example 3.1.6

For the function  $f$  of Example 3.1.3, we may write  $f(1) = a$ ,  $f(2) = b$ , and  $f(3) = a$ .

### Example 3.1.7

If we call the check digit function  $L$ , we may write

$$L(469035821375465) = 7 \quad \text{and} \quad L(469035827375465) = 4.$$

The next example shows how we sometimes use the  $f(x)$  notation to define a function.

### Example 3.1.8

Let  $f$  be the function defined by the rule  $f(x) = x^2$ . For example,  $f(2) = 4$ ,  $f(-3.5) = 12.25$ , and  $f(0) = 0$ . Although we frequently find functions defined in this way, the definition is incomplete since the domain and codomain are not specified. If we are told that the domain is the set of all real numbers and the codomain is the set of all nonnegative real numbers, in ordered-pair notation, we would have

$$f = \{(x, x^2) \mid x \text{ is a real number}\}.$$

The range of  $f$  is the set of all nonnegative real numbers.

### Example 3.1.9

Most calculators have a  $1/x$  key. If you enter a number and hit the  $1/x$  key, the reciprocal of the number entered (or an approximation to it) is displayed. This function can be defined by the rule

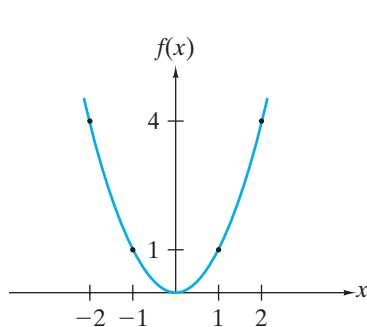
$$R(x) = \frac{1}{x}.$$

The domain is the set of all numbers that can be entered into the calculator and whose reciprocals can be computed and displayed by the calculator. The range is the set of all the reciprocals that can be computed and displayed. We could define the codomain also to be the set of all the reciprocals that can be computed and displayed. Notice that by the nature of the calculator, the domain and range are finite sets.

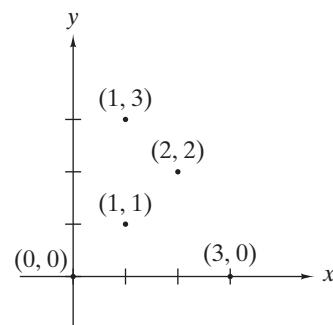
Another way to visualize a function is to draw its graph. The **graph of a function**  $f$  whose domain and codomain are subsets of the real numbers is obtained by plotting points in the plane that correspond to the elements in  $f$ . The domain is contained in the horizontal axis and the codomain is contained in the vertical axis.

**Example 3.1.10**

The graph of the function  $f(x) = x^2$  is shown in Figure 3.1.4.



**Figure 3.1.4** The graph of  $f(x) = x^2$ .



**Figure 3.1.5** A set that is not a function. The vertical line  $x = 1$  intersects two points in the set.

We note that a set  $S$  of points in the plane defines a function precisely when each vertical line intersects at most one point of  $S$ . If some vertical line contains two or more points of some set, the domain point does not assign a *unique* codomain point and the set does not define a function (see Figure 3.1.5).

Functions involving the **modulus operator** play an important role in mathematics and computer science.

**Definition 3.1.11** ► If  $x$  is an integer and  $y$  is a positive integer, we define  $x \bmod y$  to be the remainder when  $x$  is divided by  $y$ .

**Example 3.1.12**

We have

$$6 \bmod 2 = 0, \quad 5 \bmod 1 = 0, \quad 8 \bmod 12 = 8, \quad 199673 \bmod 2 = 1.$$

**Example 3.1.13**

The check digit calculated by the Luhn algorithm can be written

$$[10 - (S \bmod 10)] \bmod 10,$$

where  $S$  is the sum used in the intermediate step of the calculation. The last digit in  $S$  is given by  $S \bmod 10$ . If this digit is 1 through 9, inclusive,  $10 - (S \bmod 10)$  gives the check digit and the last “mod 10” is unnecessary, but harmless. However, if the last digit in  $S$  is 0,  $10 - (S \bmod 10) = 10$ . In this case, adding the last “mod 10” gives the check digit as 0.

**Example 3.1.14**

What day of the week will it be 365 days from Wednesday?

**SOLUTION** Seven days after Wednesday, it is Wednesday again; 14 days after Wednesday, it is Wednesday again; and in general, if  $n$  is a positive integer,  $7n$  days after Wednesday, it is Wednesday again. Thus we need to subtract as many 7's as possible from 365 and see how many days are left, which is the same as computing  $365 \bmod 7$ . Since  $365 \bmod 7 = 1$ , 365 days from Wednesday, it will be one day later, namely Thursday. This explains why, except for leap year, when an extra day is added to February, the identical month and date in consecutive years move forward one day of the week.

**Example 3.1.15****Go Online**

For more on hash functions, see  
[goo.gl/V3y4pS](http://goo.gl/V3y4pS)

**Hash Functions** Suppose that we have cells in a computer memory indexed from 0 to 10 (see Figure 3.1.6). We wish to store and retrieve arbitrary nonnegative integers in these cells. One approach is to use a **hash function**. A hash function takes a data item to be stored or retrieved and computes the first choice for a location for the item. For example, for our problem, to store or retrieve the number  $n$ , we might take as the first choice for a location,  $n \bmod 11$ . Our hash function becomes  $h(n) = n \bmod 11$ . Figure 3.1.6 shows the result of storing 15, 558, 32, 132, 102, and 5, in this order, in initially empty cells.

|     |   |   |     |    |   |     |   |     |   |    |
|-----|---|---|-----|----|---|-----|---|-----|---|----|
| 132 |   |   | 102 | 15 | 5 | 257 |   | 558 |   | 32 |
| 0   | 1 | 2 | 3   | 4  | 5 | 6   | 7 | 8   | 9 | 10 |

**Figure 3.1.6** Cells in a computer memory.

Now suppose that we want to store 257. Since  $h(257) = 4$ , then 257 should be stored at location 4; however, this position is already occupied. In this case we say that a **collision** has occurred. More precisely, a collision occurs for a hash function  $H$  if  $H(x) = H(y)$ , but  $x \neq y$ . To handle collisions, a **collision resolution policy** is required. One simple collision resolution policy is to find the next highest (with 0 assumed to follow 10) unoccupied cell. If we use this collision resolution policy, we would store 257 at location 6 (see Figure 3.1.6).

If we want to locate a stored value  $n$ , we compute  $m = h(n)$  and begin looking at location  $m$ . If  $n$  is not at this position, we look in the next-highest position (again, 0 is assumed to follow 10); if  $n$  is not in this position, we proceed to the next-highest position, and so on. If we reach an empty cell or return to our original position, we conclude that  $n$  is not present; otherwise, we obtain the position of  $n$ .

If collisions occur infrequently, and if when one does occur it is resolved quickly, then hashing provides a very fast method of storing and retrieving data. As an example, personnel data are frequently stored and retrieved by hashing on employee identification numbers.

**Example 3.1.16**

**Pseudorandom Numbers** Computers are often used to simulate random behavior. A game program might simulate rolling dice, and a client service program might simulate the arrival of customers at a bank. Such programs generate numbers that appear random and are called **pseudorandom numbers**. For example, the dice-rolling program would need pairs of pseudorandom numbers, each between 1 and 6, to simulate the outcome of rolling dice. Pseudorandom numbers are not truly random; if one knows the program that generates the numbers, one could predict what numbers would occur.

The method usually used to generate pseudorandom numbers is called the **linear congruential method**. This method requires four integers: the modulus  $m$ , the multiplier  $a$ , the increment  $c$ , and a seed  $s$  satisfying  $2 \leq a < m$ ,  $0 \leq c < m$ , and  $0 \leq s < m$ . We then set  $x_0 = s$ . The sequence of pseudorandom numbers generated,  $x_1, x_2, \dots$ , is given by the formula

$$x_n = (ax_{n-1} + c) \bmod m.$$

The formula computes the next pseudorandom number using its immediate predecessor. For example, if  $m = 11$ ,  $a = 7$ ,  $c = 5$ , and  $s = 3$ , then

$$x_1 = (ax_0 + c) \bmod m = (7 \cdot 3 + 5) \bmod 11 = 4$$

and

$$x_2 = (ax_1 + c) \bmod m = (7 \cdot 4 + 5) \bmod 11 = 0.$$

Similar computations show that the sequence continues:

$$x_3 = 5, x_4 = 7, x_5 = 10, x_6 = 9, x_7 = 2, x_8 = 8, x_9 = 6, x_{10} = 3.$$

Since  $x_{10} = 3$ , which is the value of the seed, the sequence now repeats: 3, 4, 0, 5, 7, ... .

Much effort has been invested in finding good values for a linear congruential method. Critical simulations such as those involving aircraft and nuclear research require “good” random numbers. In practice, large values are used for  $m$  and  $a$ . Commonly used values are  $m = 2^{31} - 1 = 2,147,483,647$ ;  $a = 7^5 = 16,807$ ; and  $c = 0$ , which generate a sequence of  $2^{31} - 1$  integers before repeating a value. ▲

In the 1990s, Daniel Corriveau of Quebec won three straight games of a computer keno game in Montreal, each time choosing 19 of 20 numbers correctly. The odds against this feat are 6 billion to 1. Suspicious officials at first refused to pay him. Although Corriveau attributed his success to chaos theory, what in fact happened was that whenever power was cut, the random number generator started with the same seed, thus generating the same sequence of numbers. The embarrassed casino finally paid Corriveau the \$600,000 due him.

We next define the **floor** and **ceiling** of a real number.

**Definition 3.1.17** ► The *floor* of  $x$ , denoted  $\lfloor x \rfloor$ , is the greatest integer less than or equal to  $x$ . The *ceiling* of  $x$ , denoted  $\lceil x \rceil$ , is the least integer greater than or equal to  $x$ . ▲

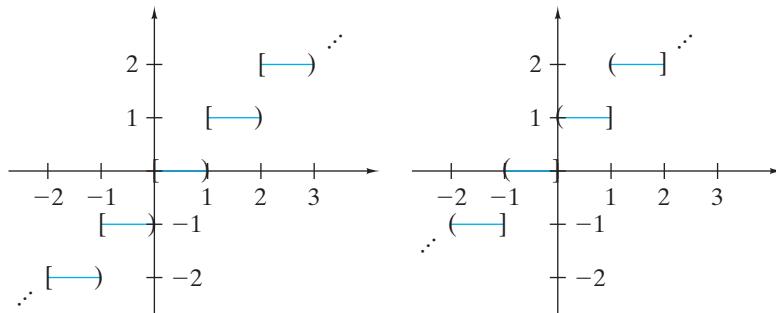
### Example 3.1.18

$$\lfloor 8.3 \rfloor = 8, \lceil 9.1 \rceil = 10, \lfloor -8.7 \rfloor = -9, \lceil -11.3 \rceil = -11, \lfloor 6 \rfloor = 6, \lceil -8 \rceil = -8$$
 ▲

The floor of  $x$  “rounds  $x$  down” while the ceiling of  $x$  “rounds  $x$  up.” We will use the floor and ceiling functions throughout the book.

### Example 3.1.19

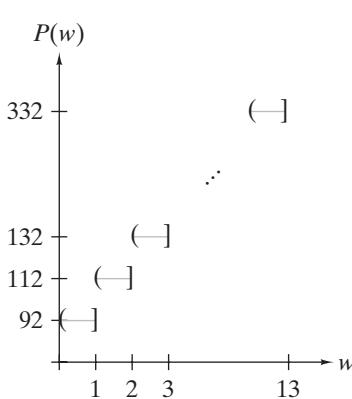
Figure 3.1.7 shows the graphs of the floor and ceiling functions. A bracket, [ or ], indicates that the point is to be included in the graph; a parenthesis, ( or ), indicates that the point is to be excluded from the graph.



**Figure 3.1.7** The graphs of the floor (left graph) and ceiling (right graph) functions. ▲

### Example 3.1.20

The first-class postage rate for mail up to 13 ounces is 92 cents for the first ounce or fraction thereof and 20 cents for each additional ounce or fraction thereof. The postage  $P(w)$  as a function of weight  $w$  is given by the equation



**Figure 3.1.8** The graph of the postage function  $P(w) = 92 + 20\lceil w - 1 \rceil$ .

$$P(w) = 92 + 20\lceil w - 1 \rceil \quad 13 \geq w > 0.$$

The expression  $\lceil w - 1 \rceil$  counts the number of additional ounces beyond 1, with a fraction counting as one additional ounce. As examples,

$$\begin{aligned} P(3.7) &= 92 + 20\lceil 3.7 - 1 \rceil = 92 + 20\lceil 2.7 \rceil = 92 + 20 \cdot 3 = 152, \\ P(2) &= 92 + 20\lceil 2 - 1 \rceil = 92 + 20\lceil 1 \rceil = 92 + 20 \cdot 1 = 112. \end{aligned}$$

The graph of the function  $P$  is shown in Figure 3.1.8. ▲

The Quotient-Remainder Theorem (Theorem 2.5.6) states that if  $d$  and  $n$  are integers,  $d > 0$ , there exist integers  $q$  (quotient) and  $r$  (remainder) satisfying

$$n = dq + r \quad 0 \leq r < d.$$

Dividing by  $d$ , we obtain

$$\frac{n}{d} = q + \frac{r}{d}.$$

Since  $0 \leq r/d < 1$ ,

$$\left\lfloor \frac{n}{d} \right\rfloor = \left\lfloor q + \frac{r}{d} \right\rfloor = q.$$

Thus, we may compute the quotient  $q$  as  $\lfloor n/d \rfloor$ . Having computed the quotient  $q$ , we may compute the remainder as  $r = n - dq$ . We previously introduced the notation  $n \bmod d$  for the remainder.

### Example 3.1.21

We have  $36844/2427 = 15.18088\dots$ ; thus the quotient is  $q = \lfloor 36844/2427 \rfloor = 15$ . Therefore, the remainder  $36844 \bmod 2427$  is  $r = 36844 - 2427 \cdot 15 = 439$ . We have  $n = dq + r$  or  $36844 = 2427 \cdot 15 + 439$ . ▲

**Definition 3.1.22** ► A function  $f$  from  $X$  to  $Y$  is said to be *one-to-one* (or *injective*) if for all  $x_1, x_2 \in X$ , if  $f(x_1) = f(x_2)$  then  $x_1 = x_2$ . ▲

An equivalent way to state Definition 3.1.22 is: If  $y$  is an element of the range of  $f$ , then there is *exactly one*  $x$  in the domain of  $f$  such that  $f(x) = y$ . If there were two distinct elements  $x_1$  and  $x_2$  of the domain of  $f$  with  $f(x_1) = y = f(x_2)$ , then we would have  $f(x_1) = f(x_2)$  but  $x_1 \neq x_2$ —a counterexample to the claim that  $f$  is one-to-one.

Because the amount of potential data is usually so much larger than the available memory, hash functions are usually not one-to-one (see Example 3.1.15). In other words, most hash functions produce collisions.

### Example 3.1.23

The function  $f = \{(1, b), (3, a), (2, c)\}$  from  $X = \{1, 2, 3\}$  to  $Y = \{a, b, c, d\}$  is one-to-one. ▲

### Example 3.1.24

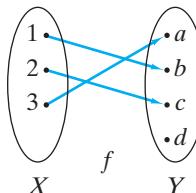
The function  $f = \{(1, a), (2, b), (3, a)\}$  is not one-to-one since  $f(1) = a = f(3)$ . ▲

### Example 3.1.25

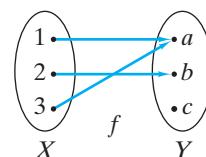
If  $X$  is the set of persons who have social security numbers and we assign each person  $x \in X$  his or her social security number  $SS(x)$ , we obtain a one-to-one function since distinct persons are always assigned distinct social security numbers. It is because this correspondence is one-to-one that the government uses social security numbers as identifiers. ▲

**Example 3.1.26**

If a function from  $X$  to  $Y$  is one-to-one, each element in  $Y$  in its arrow diagram will have at most one arrow pointing to it (see Figure 3.1.9). If a function is not one-to-one, some element in  $Y$  in its arrow diagram will have two or more arrows pointing to it (see Figure 3.1.10).



**Figure 3.1.9** The function of Example 3.1.23. This function is one-to-one because each element in  $Y$  has at most one arrow pointing to it. This function is not onto  $Y$  because there is no arrow pointing to  $d$ .



**Figure 3.1.10** A function that is not one-to-one. This function is not one-to-one because  $a$  has two arrows pointing to it. This function is not onto  $Y$  because there is no arrow pointing to  $c$ .

**Example 3.1.27**

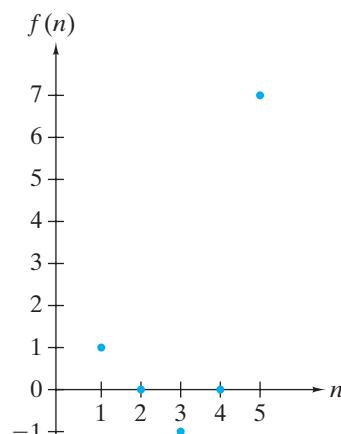
Prove that the function  $f(n) = 2n + 1$  from the set of positive integers to the set of positive integers is one-to-one.

**SOLUTION** We must show that for all positive integers  $n_1$  and  $n_2$ , if  $f(n_1) = f(n_2)$ , then  $n_1 = n_2$ . So, suppose that  $f(n_1) = f(n_2)$ . Using the definition of  $f$ , this latter equation translates as  $2n_1 + 1 = 2n_2 + 1$ . Subtracting 1 from both sides of the equation and then dividing both sides of the equation by 2 yields  $n_1 = n_2$ . Therefore,  $f$  is one-to-one. ▲

**Example 3.1.28**

Prove that the function  $f(n) = 2^n - n^2$  from the set of positive integers to the set of integers is *not* one-to-one.

**SOLUTION** We must find positive integers  $n_1$  and  $n_2$ ,  $n_1 \neq n_2$ , such that  $f(n_1) = f(n_2)$ . By checking the graph (see Figure 3.1.11) or otherwise, we find that  $f(2) = f(4)$ . Therefore,  $f$  is not one-to-one.



**Figure 3.1.11** The graph of  $f(n) = 2^n - n^2$ .

If the range of a function  $f$  is equal to its codomain  $Y$ , the function is said to be **onto**  $Y$ .

**Definition 3.1.29** ► A function  $f$  from  $X$  to  $Y$  is said to be *onto*  $Y$  (or *surjective*) if for every  $y \in Y$ , there exists  $x \in X$  such that  $f(x) = y$ .

**Example 3.1.30**

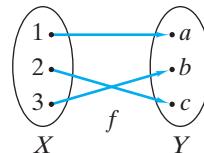
The function  $f = \{(1, a), (2, c), (3, b)\}$  from  $X = \{1, 2, 3\}$  to  $Y = \{a, b, c\}$  is one-to-one and onto  $Y$ .

**Example 3.1.31**

The function  $f = \{(1, b), (3, a), (2, c)\}$  from  $X = \{1, 2, 3\}$  to  $Y = \{a, b, c, d\}$  is *not* onto  $Y$ .

**Example 3.1.32**

If a function from  $X$  to  $Y$  is onto, each element in  $Y$  in its arrow diagram will have at least one arrow pointing to it (see Figure 3.1.12). If a function from  $X$  to  $Y$  is not onto, some element in  $Y$  in its arrow diagram will fail to have an arrow pointing to it (see Figures 3.1.9 and 3.1.10).



**Figure 3.1.12** The function of Example 3.1.30. This function is one-to-one because each element in  $Y$  has at most one arrow. This function is onto because each element in  $Y$  has at least one arrow pointing to it.

**Example 3.1.33**

Prove that the function

$$f(x) = \frac{1}{x^2}$$

from the set  $X$  of nonzero real numbers to the set  $Y$  of positive real numbers is onto  $Y$ .

**SOLUTION** We must show that for every  $y \in Y$ , there exists  $x \in X$  such that  $f(x) = y$ . Substituting the formula for  $f(x)$ , this last equation becomes

$$\frac{1}{x^2} = y.$$

Solving for  $x$ , we find

$$x = \pm \frac{1}{\sqrt{y}}.$$

Notice that  $1/\sqrt{y}$  is defined because  $y$  is a positive real number. If we take  $x$  to be the positive square root

$$x = \frac{1}{\sqrt{y}},$$

then  $x \in X$ . (We could just as well have taken  $x = -1/\sqrt{y}$ .) Thus, for every  $y \in Y$ , there exists  $x$ , namely,  $x = 1/\sqrt{y}$  such that

$$f(x) = f(1/\sqrt{y}) = \frac{1}{(1/\sqrt{y})^2} = y.$$

Therefore,  $f$  is onto  $Y$ . ◀

A function  $f$  from  $X$  to  $Y$  is *not* onto  $Y$  if for some  $y \in Y$ , for every  $x \in X$ ,  $f(x) \neq y$ . In other words,  $y$  is a counterexample to the claim that for every  $y \in Y$ , there exists  $x \in X$  such that  $f(x) = y$ .

### Example 3.1.34

Prove that the function  $f(n) = 2n - 1$  from the set  $X$  of positive integers to the set  $Y$  of positive integers is *not* onto  $Y$ .

**SOLUTION** We must find an element  $m \in Y$  such that for all  $n \in X$ ,  $f(n) \neq m$ . Since  $f(n)$  is an odd integer for all  $n$ , we may choose for  $y$  any positive, even integer, for example,  $y = 2$ . Then  $y \in Y$  and  $f(n) \neq y$  for all  $n \in X$ . Thus  $f$  is not onto  $Y$ . ◀

**Definition 3.1.35** ► A function that is both one-to-one and onto is called a *bijection*. ◀

### Example 3.1.36

The function  $f$  of Example 3.1.30 is a bijection. ◀

### Example 3.1.37

If  $f$  is a bijection from a finite set  $X$  to a finite set  $Y$ , then  $|X| = |Y|$ , that is, the sets have the same cardinality and are the same size. For example,  $f = \{(1, a), (2, b), (3, c), (4, d)\}$  is a bijection from  $X = \{1, 2, 3, 4\}$  to  $Y = \{a, b, c, d\}$ . Both sets have four elements. In effect,  $f$  counts the elements in  $Y$ :  $f(1) = a$  is the first element in  $Y$ ;  $f(2) = b$  is the second element in  $Y$ ; and so on. ◀

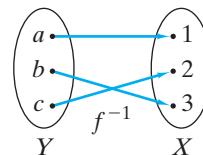
Suppose that  $f$  is a one-to-one, onto function from  $X$  to  $Y$ . It can be shown (see Exercise 116) that  $\{(y, x) \mid (x, y) \in f\}$  is a one-to-one, onto function from  $Y$  to  $X$ . This new function, denoted  $f^{-1}$ , is called  $f$  **inverse**.

### Example 3.1.38

For the function  $f = \{(1, a), (2, c), (3, b)\}$ , we have  $f^{-1} = \{(a, 1), (c, 2), (b, 3)\}$ . ◀

### Example 3.1.39

Given the arrow diagram for a one-to-one, onto function  $f$  from  $X$  to  $Y$ , we can obtain the arrow diagram for  $f^{-1}$  simply by reversing the direction of each arrow (see Figure 3.1.13, which is the arrow diagram for  $f^{-1}$ , where  $f$  is the function of Figure 3.1.12).



**Figure 3.1.13** The inverse of the function in Figure 3.1.12. The inverse is obtained by reversing all of the arrows in Figure 3.1.12. ◀

### Example 3.1.40

The function  $f(x) = 2^x$  is a one-to-one function from the set  $\mathbf{R}$  of all real numbers onto the set  $\mathbf{R}^+$  of all positive real numbers. Derive a formula for  $f^{-1}(y)$ .

**SOLUTION** Suppose that  $(y, x)$  is in  $f^{-1}$ ; that is,

$$f^{-1}(y) = x. \quad (3.1.3)$$

Then  $(x, y) \in f$ . Thus,  $y = 2^x$ . By the definition of logarithm,

$$\log_2 y = x. \quad (3.1.4)$$

Combining (3.1.3) and (3.1.4), we have  $f^{-1}(y) = x = \log_2 y$ . That is, for each  $y \in \mathbf{R}^+$ ,  $f^{-1}(y)$  is the logarithm to the base 2 of  $y$ . We can summarize the situation by saying that the inverse of the exponential function is the logarithm function. ▲

Let  $g$  be a function from  $X$  to  $Y$  and let  $f$  be a function from  $Y$  to  $Z$ . Given  $x \in X$ , we may apply  $g$  to determine a unique element  $y = g(x) \in Y$ . We may then apply  $f$  to determine a unique element  $z = f(y) = f(g(x)) \in Z$ . This compound action is called **composition**.

**Definition 3.1.41** ► Let  $g$  be a function from  $X$  to  $Y$  and let  $f$  be a function from  $Y$  to  $Z$ . The *composition of  $f$  with  $g$* , denoted  $f \circ g$ , is the function

$$(f \circ g)(x) = f(g(x))$$

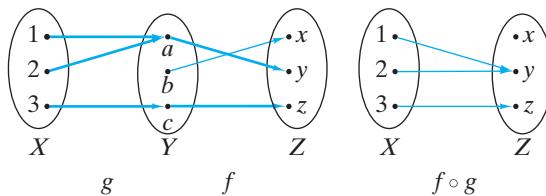
from  $X$  to  $Z$ . ▲

#### Example 3.1.42

Given  $g = \{(1, a), (2, a), (3, c)\}$ , a function from  $X = \{1, 2, 3\}$  to  $Y = \{a, b, c\}$ , and  $f = \{(a, y), (b, x), (c, z)\}$ , a function from  $Y$  to  $Z = \{x, y, z\}$ , the composition function from  $X$  to  $Z$  is the function  $f \circ g = \{(1, y), (2, y), (3, z)\}$ . ▲

#### Example 3.1.43

Given the arrow diagram for a function  $g$  from  $X$  to  $Y$  and the arrow diagram for a function  $f$  from  $Y$  to  $Z$ , we can obtain the arrow diagram for the composition  $f \circ g$  simply by “following the arrows” (see Figure 3.1.14).



**Figure 3.1.14** The composition of the functions of Example 3.1.42. The composition is obtained by drawing an arrow from  $x$  in  $X$  to  $z$  in  $Z$  provided that there are arrows from  $x$  to some  $y$  in  $Y$  and from  $y$  to  $z$ . ▲

#### Example 3.1.44

If  $f(x) = \log_3 x$  and  $g(x) = x^4$ , then  $f(g(x)) = \log_3(x^4)$ , and  $g(f(x)) = (\log_3 x)^4$ . ▲

#### Example 3.1.45

A store offers 15% off the price of certain items. A coupon is also available that offers \$20 off the price of the same items. The store will honor both discounts. The function  $D(p) = 0.85p$  gives the cost with 15% off the price  $p$ . The function  $C(p) = p - 20$  gives the cost using the \$20 coupon. The composition

$$(D \circ C)(p) = 0.85(p - 20) = 0.85p - 17$$

gives the cost using first the coupon and then the 15% discount. The composition  $(C \circ D)(p) = 0.85p - 20$  gives the cost using first the 15% discount and then the coupon.

We see that regardless of the price of an item, it is always cheapest to use the discount first.

### Example 3.1.46

Composition sometimes allows us to decompose complicated functions into simpler functions. For example, the function  $f(x) = \sqrt{\sin 2x}$  can be decomposed into the functions

$$g(x) = \sqrt{x}, \quad h(x) = \sin x, \quad w(x) = 2x.$$

We can then write  $f(x) = g(h(w(x)))$ . This decomposition technique is important in differential calculus since there are rules for differentiating simple functions such as  $g$ ,  $h$ , and  $w$  and also rules about how to differentiate the composition of functions. Combining these rules, we can differentiate more complicated functions.

A **binary operator** on a set  $X$  associates with each ordered pair of elements in  $X$  one element in  $X$ .

**Definition 3.1.47** ► A function from  $X \times X$  to  $X$  is called a *binary operator* on  $X$ .

### Example 3.1.48

Let  $X = \{1, 2, \dots\}$ . If we define  $f(x, y) = x + y$ , where  $x, y \in X$ , then  $f$  is a binary operator on  $X$ .

### Example 3.1.49

If  $X$  is a set of propositions,  $\wedge$ ,  $\vee$ ,  $\rightarrow$ , and  $\leftrightarrow$  are binary operators on  $X$ .

A **unary operator** on a set  $X$  associates with each single element of  $X$  one element in  $X$ .

**Definition 3.1.50** ► A function from  $X$  to  $X$  is called a *unary operator* on  $X$ .

### Example 3.1.51

Let  $U$  be a universal set. If we define  $f(X) = \bar{X}$ , where  $X \in \mathcal{P}(U)$ , then  $f$  is a unary operator on  $\mathcal{P}(U)$ .

### Example 3.1.52

If  $X$  is a set of propositions,  $\neg$  is a unary operator on  $X$ .

## 3.1 Problem-Solving Tips

The key to solving problems involving functions is clearly understanding the definition of function. A function  $f$  from  $X$  to  $Y$  can be thought of in many ways. Formally,  $f$  is a subset of  $X \times Y$  having the property that for every  $x \in X$ , there is a unique  $y \in Y$  such that  $(x, y) \in X \times Y$ . Informally,  $f$  can be thought of as a *mapping* of elements from  $X$  to  $Y$ . The arrow diagram emphasizes this view of a function. For an arrow diagram to be a function, there must be exactly one arrow from each element in  $X$  to some element in  $Y$ .

A function is a very general concept. Any subset of  $X \times Y$  having the property that for every  $x \in X$ , there is a unique  $y \in Y$  such that  $(x, y) \in X \times Y$  is a function. A function may be defined by listing its members; for example,  $\{(a, 1), (b, 3), (c, 2), (d, 1)\}$  is a function from  $\{a, b, c, d\}$  to  $\{1, 2, 3\}$ . Here, there is apparently no formula for membership; the definition just tells us which pairs make up the function.

On the other hand, a function may be defined by a formula. For example,

$$\{(n, n + 2) \mid n \text{ is a positive integer}\}$$

defines a function from the set of positive integers to the set of positive integers. The “formula” for the mapping is “add 2.”

The  $f(x)$  notation may be used to indicate which element in the codomain is associated with an element  $x$  in the domain or to define a function. For example, for the function  $f = \{(a, 1), (b, 3), (c, 2), (d, 1)\}$ , we could write  $f(a) = 1, f(b) = 3$ , and so on. Assuming that the domain of definition is the positive integers, the equation  $g(n) = n+2$  defines the function  $\{(n, n+2) \mid n \text{ is a positive integer}\}$  from the set of positive integers to the set of positive integers.

To prove that a function  $f$  from  $X$  to  $Y$  is one-to-one, show that for all  $x_1, x_2 \in X$ , if  $f(x_1) = f(x_2)$ , then  $x_1 = x_2$ .

To prove that a function  $f$  from  $X$  to  $Y$  is *not* one-to-one, find  $x_1, x_2 \in X, x_1 \neq x_2$ , such that  $f(x_1) = f(x_2)$ .

To prove that a function  $f$  from  $X$  to  $Y$  is onto, show that for all  $y \in Y$ , there exists  $x \in X$  such that  $f(x) = y$ .

To prove that a function  $f$  from  $X$  to  $Y$  is *not* onto, find  $y \in Y$  such that  $f(x) \neq y$  for all  $x \in X$ .

### 3.1 Review Exercises

1. What is a function from  $X$  to  $Y$ ?
2. Explain how to use an arrow diagram to depict a function.
3. What is the graph of a function?
4. Given a set of points in the plane, how can we tell whether it is a function?
5. What is the value of  $x \bmod y$ ?
6. What is a hash function?
7. What is a collision for a hash function?
8. What is a collision resolution policy?
9. What are pseudorandom numbers?
10. Explain how a linear congruential random number generator works, and give an example of a linear congruential random number generator.
11. What is the floor of  $x$ ? How is the floor denoted?
12. What is the ceiling of  $x$ ? How is the ceiling denoted?
13. Define *one-to-one function*. Give an example of a one-to-one function. Explain how to use an arrow diagram to determine whether a function is one-to-one.
14. Define *onto function*. Give an example of an onto function. Explain how to use an arrow diagram to determine whether a function is onto.
15. What is a bijection? Give an example of a bijection. Explain how to use an arrow diagram to determine whether a function is a bijection.
16. Define *inverse function*. Give an example of a function and its inverse. Given the arrow diagram of a function, how can we find the arrow diagram of the inverse function?
17. Define *composition of functions*. How is the composition of  $f$  and  $g$  denoted? Give an example of functions  $f$  and  $g$  and their composition. Given the arrow diagrams of two functions, how can we find the arrow diagram of the composition of the functions?
18. What is a binary operator? Give an example of a binary operator.
19. What is a unary operator? Give an example of a unary operator.

### 3.1 Exercises

In Exercises 1–6, determine which credit card numbers have correct check digits.

1. 5366-2806-9965-4138
2. 5194-1132-8860-3905
3. 4004-6067-3429-0019
4. 3419-6888-7169-444

5. 3016-4773-7532-21
6. 4629-9521-3698-0203
7. Show that when 82 in the valid credit card number 4690-3582-1375-4657 is transposed to 28, the check digit changes.

Determine whether each set in Exercises 8–12 is a function from  $X = \{1, 2, 3, 4\}$  to  $Y = \{a, b, c, d\}$ . If it is a function, find its do-

main and range, draw its arrow diagram, and determine if it is one-to-one, onto, or both. If it is both one-to-one and onto, give the description of the inverse function as a set of ordered pairs, draw its arrow diagram, and give the domain and range of the inverse function.

8.  $\{(1, a), (2, a), (3, c), (4, b)\}$
9.  $\{(1, c), (2, a), (3, b), (4, c), (2, d)\}$
10.  $\{(1, c), (2, d), (3, a), (4, b)\}$
11.  $\{(1, d), (2, d), (4, a)\}$
12.  $\{(1, b), (2, b), (3, b), (4, b)\}$

Draw the graphs of the functions in Exercises 13–16. The domain of each function is the set of real numbers. The codomain of each function is also the set of real numbers.

13.  $f(x) = \lceil x \rceil - \lfloor x \rfloor$
14.  $f(x) = x - \lfloor x \rfloor$
15.  $f(x) = \lceil x^2 \rceil$
16.  $f(x) = \lfloor x^2 - x \rfloor$

Determine whether each function in Exercises 17–22 is one-to-one, onto, or both. Prove your answers. The domain of each function is the set of all integers. The codomain of each function is also the set of all integers.

17.  $f(n) = n + 1$
18.  $f(n) = n^2 - 1$
19.  $f(n) = \lceil n/2 \rceil$
20.  $f(n) = |n|$
21.  $f(n) = 2n$
22.  $f(n) = n^3$

Determine whether each function in Exercises 23–28 is one-to-one, onto, or both. Prove your answers. The domain of each function is  $\mathbf{Z} \times \mathbf{Z}$ . The codomain of each function is  $\mathbf{Z}$ .

23.  $f(m, n) = m - n$
24.  $f(m, n) = m$
25.  $f(m, n) = mn$
26.  $f(m, n) = m^2 + n^2$
27.  $f(m, n) = n^2 + 1$
28.  $f(m, n) = m + n + 2$

29. Prove that the function  $f$  from  $\mathbf{Z}^+ \times \mathbf{Z}^+$  to  $\mathbf{Z}^+$  defined by  $f(m, n) = 2^m 3^n$  is one-to-one but not onto.

Determine whether each function in Exercises 30–35 is one-to-one, onto, or both. Prove your answers. The domain of each function is the set of all real numbers. The codomain of each function is also the set of all real numbers.

30.  $f(x) = 6x - 9$
31.  $f(x) = 3x^2 - 3x + 1$
32.  $f(x) = \sin x$
33.  $f(x) = 2x^3 - 4$
34.  $f(x) = 3^x - 2$
35.  $f(x) = \frac{x}{1+x^2}$

36. Give an example of a function different from those presented in the text that is one-to-one but not onto, and prove that your function has the required properties.
37. Give an example of a function different from those presented in the text that is onto but not one-to-one, and prove that your function has the required properties.

38. Give an example of a function different from those presented in the text that is neither one-to-one nor onto, and prove that your function has the required properties.
39. Write the definition of “one-to-one” using logical notation (i.e., use  $\forall, \exists$ , etc.).
40. Use De Morgan’s laws of logic to negate the definition of “one-to-one.”
41. Write the definition of “onto” using logical notation (i.e., use  $\forall, \exists$ , etc.).
42. Use De Morgan’s laws of logic to negate the definition of “onto.”

Each function in Exercises 43–48 is one-to-one on the specified domain  $X$ . By letting  $Y = \text{range of } f$ , we obtain a bijection from  $X$  to  $Y$ . Find each inverse function.

43.  $f(x) = 4x + 2, X = \text{set of real numbers}$
44.  $f(x) = 3^x, X = \text{set of real numbers}$
45.  $f(x) = 3 \log_2 x, X = \text{set of positive real numbers}$
46.  $f(x) = 3 + \frac{1}{x}, X = \text{set of nonzero real numbers}$
47.  $f(x) = 4x^3 - 5, X = \text{set of real numbers}$
48.  $f(x) = 6 + 2^{7x-1}, X = \text{set of real numbers}$
49. Given

$$g = \{(1, b), (2, c), (3, a)\},$$

a function from  $X = \{1, 2, 3\}$  to  $Y = \{a, b, c, d\}$ , and

$$f = \{(a, x), (b, x), (c, z), (d, w)\},$$

a function from  $Y$  to  $Z = \{w, x, y, z\}$ , write  $f \circ g$  as a set of ordered pairs and draw the arrow diagram of  $f \circ g$ .

50. Let  $f$  and  $g$  be functions from the positive integers to the positive integers defined by the equations

$$f(n) = 2n + 1, \quad g(n) = 3n - 1.$$

Find the compositions  $f \circ f$ ,  $g \circ g$ ,  $f \circ g$ , and  $g \circ f$ .

51. Let  $f$  and  $g$  be functions from the positive integers to the positive integers defined by the equations

$$f(n) = n^2, \quad g(n) = 2^n.$$

Find the compositions  $f \circ f$ ,  $g \circ g$ ,  $f \circ g$ , and  $g \circ f$ .

52. Let  $f$  and  $g$  be functions from the nonnegative real numbers to the nonnegative real numbers defined by the equations

$$f(x) = \lfloor 2x \rfloor, \quad g(x) = x^2.$$

Find the compositions  $f \circ f$ ,  $g \circ g$ ,  $f \circ g$ , and  $g \circ f$ .

53. A store offers a (fixed, nonzero) percentage off the price of certain items. A coupon is also available that offers a (fixed, nonzero) amount off the price of the same items. The store will honor both discounts. Show that regardless of the price of an item, the percentage off the price, and amount off the price, it is always cheapest to use the coupon first.

In Exercises 54–59, decompose the function into simpler functions as in Example 3.1.46.

**54.**  $f(x) = \log_2(x^2 + 2)$

**56.**  $f(x) = \sin 2x$

**58.**  $f(x) = (3 + \sin x)^4$

**60.** Given

$$f = \{(x, x^2) \mid x \in X\},$$

a function from  $X = \{-5, -4, \dots, 4, 5\}$  to the set of integers, write  $f$  as a set of ordered pairs and draw the arrow diagram of  $f$ . Is  $f$  one-to-one or onto?

- 61.** How many functions are there from  $\{1, 2\}$  to  $\{a, b\}$ ? Which are one-to-one? Which are onto?

**62.** Given

$$f = \{(a, b), (b, a), (c, b)\},$$

a function from  $X = \{a, b, c\}$  to  $X$ :

- (a) Write  $f \circ f$  and  $f \circ f \circ f$  as sets of ordered pairs.  
 (b) Define

$$f^n = f \circ f \circ \dots \circ f$$

to be the  $n$ -fold composition of  $f$  with itself. Write  $f^0$  and  $f^{623}$  as sets of ordered pairs.

- 63.** Let  $f$  be the function from  $X = \{0, 1, 2, 3, 4\}$  to  $X$  defined by

$$f(x) = 4x \bmod 5.$$

Write  $f$  as a set of ordered pairs and draw the arrow diagram of  $f$ . Is  $f$  one-to-one? Is  $f$  onto?

- 64.** Let  $f$  be the function from  $X = \{0, 1, 2, 3, 4, 5\}$  to  $X$  defined by

$$f(x) = 4x \bmod 6.$$

Write  $f$  as a set of ordered pairs and draw the arrow diagram of  $f$ . Is  $f$  one-to-one? Is  $f$  onto?

- 65.** An International Standard Book Number (ISBN) is a code of 13 characters separated by dashes, such as 978-1-59448-950-1. An ISBN consists of five parts: a product code, a group code, a publisher code, a code that uniquely identifies the book among those published by the particular publisher, and a check digit. For 978-1-59448-950-1, the product code 978 identifies the product as a book (the same scheme is used for other products). The group code is 1, which identifies the book as one from an English-speaking country. The publisher code 59448 identifies the book as one published by Riverhead Books, Penguin Group. The code 950 uniquely identifies the book among those published by Riverhead Books, Penguin Group (*Hosseini: A Thousand Splendid Suns*, in this case). The check digit is 1.

Let  $S$  equal the sum of the first digit, plus three times the second digit, plus the third digit, plus three times the

fourth digit, ..., plus three times the twelfth digit. The check digit is equal to

$$[10 - (S \bmod 10)] \bmod 10.$$

Universal Product Codes (UPC), the bar codes that are scanned at the grocery store for example, use a similar method to compute the check digit.

Verify the check digit for this book.

In Exercises 66–71, determine which ISBNs (see Exercise 65) have correct check digits.

**66.** 978-1-61374-376-9

**67.** 978-0-8108-8139-2

**68.** 978-0-939460-91-5

**69.** 978-0-8174-3593-6

**70.** 978-1-4354-6028-7

**71.** 978-0-684-87018-0

- 72.** Show that if a single digit of an ISBN is changed, the check digit will change. Thus, any single-digit error will be detected.

For each hash function in Exercises 73–76, show how the data would be inserted in the order given in initially empty cells. Use the collision resolution policy of Example 3.1.15.

**73.**  $h(x) = x \bmod 11$ ; cells indexed 0 to 10; data: 53, 13, 281, 743, 377, 20, 10, 796

**74.**  $h(x) = x \bmod 17$ ; cells indexed 0 to 16; data: 714, 631, 26, 373, 775, 906, 509, 2032, 42, 4, 136, 1028

**75.**  $h(x) = x^2 \bmod 11$ ; cells and data as in Exercise 73

**76.**  $h(x) = (x^2 + x) \bmod 17$ ; cells and data as in Exercise 74

- 77.** Suppose that we store and retrieve data as described in Example 3.1.15. Will any problem arise if we delete data? Explain.

- 78.** Suppose that we store data as described in Example 3.1.15 and that we never store more than 10 items. Will any problem arise when retrieving data if we stop searching when we encounter an empty cell? Explain.

- 79.** Suppose that we store data as described in Example 3.1.15 and retrieve data as described in Exercise 78. Will any problem arise if we delete data? Explain.

Let  $g$  be a function from  $X$  to  $Y$  and let  $f$  be a function from  $Y$  to  $Z$ . For each statement in Exercises 80–87, if the statement is true, prove it; otherwise, give a counterexample.

- 80.** If  $g$  is one-to-one, then  $f \circ g$  is one-to-one.

- 81.** If  $f$  is onto, then  $f \circ g$  is onto.

- 82.** If  $g$  is onto, then  $f \circ g$  is onto.

- 83.** If  $f$  and  $g$  are onto, then  $f \circ g$  is onto.

- 84.** If  $f$  and  $g$  are one-to-one and onto, then  $f \circ g$  is one-to-one and onto.

- 85.** If  $f \circ g$  is one-to-one, then  $f$  is one-to-one.

- 86.** If  $f \circ g$  is one-to-one, then  $g$  is one-to-one.

- 87.** If  $f \circ g$  is onto, then  $f$  is onto.

If  $f$  is a function from  $X$  to  $Y$  and  $A \subseteq X$  and  $B \subseteq Y$ , we define

$$f(A) = \{f(x) \mid x \in A\}, \quad f^{-1}(B) = \{x \in X \mid f(x) \in B\}.$$

We call  $f^{-1}(B)$  the inverse image of  $B$  under  $f$ .

**88.** Let

$$g = \{(1, a), (2, c), (3, c)\}$$

be a function from  $X = \{1, 2, 3\}$  to  $Y = \{a, b, c, d\}$ . Let  $S = \{1\}$ ,  $T = \{1, 3\}$ ,  $U = \{a\}$ , and  $V = \{a, c\}$ . Find  $g(S)$ ,  $g(T)$ ,  $g^{-1}(U)$ , and  $g^{-1}(V)$ .

**★89.** Let  $f$  be a function from  $X$  to  $Y$ . Prove that  $f$  is one-to-one if and only if

$$f(A \cap B) = f(A) \cap f(B)$$

for all subsets  $A$  and  $B$  of  $X$ . [When  $S$  is a set, we define  $f(S) = \{f(x) \mid x \in S\}$ .]

**★90.** Let  $f$  be a function from  $X$  to  $Y$ . Prove that  $f$  is one-to-one if and only if whenever  $g$  is a one-to-one function from any set  $A$  to  $X$ ,  $f \circ g$  is one-to-one.

**★91.** Let  $f$  be a function from  $X$  to  $Y$ . Prove that  $f$  is onto  $Y$  if and only if whenever  $g$  is a function from  $Y$  onto any set  $Z$ ,  $g \circ f$  is onto  $Z$ .

**92.** Let  $f$  be a function from  $X$  onto  $Y$ . Let

$$\mathcal{S} = \{f^{-1}(\{y\}) \mid y \in Y\}.$$

Show that  $\mathcal{S}$  is a partition of  $X$ .

Let  $\mathbf{R}^{\mathbf{R}}$  denote the set of functions from  $\mathbf{R}$  to  $\mathbf{R}$ . We define the evaluation function  $E_a$ , where  $a \in \mathbf{R}$ , from  $\mathbf{R}^{\mathbf{R}}$  to  $\mathbf{R}$  as

$$E_a(f) = f(a).$$

**93.** Is  $E_1$  one-to-one? Prove your answer.

**94.** Is  $E_1$  onto? Prove your answer.

**95.** Let  $f$  be a function from  $\mathbf{R}$  to  $\mathbf{R}$  such that for some  $r \in \mathbf{R}$ ,  $f(rx) = rf(x)$  for all  $x \in \mathbf{R}$ . Prove that  $f(r^n x) = r^n f(x)$  for all  $x \in \mathbf{R}$ ,  $n \in \mathbf{Z}^+$ .

Exercises 96–100 use the following definitions. Let  $X = \{a, b, c\}$ . Define a function  $S$  from  $\mathcal{P}(X)$  to the set of bit strings of length 3 as follows. Let  $Y \subseteq X$ . If  $a \in Y$ , set  $s_1 = 1$ ; if  $a \notin Y$ , set  $s_1 = 0$ . If  $b \in Y$ , set  $s_2 = 1$ ; if  $b \notin Y$ , set  $s_2 = 0$ . If  $c \in Y$ , set  $s_3 = 1$ ; if  $c \notin Y$ , set  $s_3 = 0$ . Define  $S(Y) = s_1s_2s_3$ .

**96.** What is the value of  $S(\{a, c\})$ ?

**97.** What is the value of  $S(\emptyset)$ ?

**98.** What is the value of  $S(X)$ ?

**99.** Prove that  $S$  is one-to-one.

**100.** Prove that  $S$  is onto.

Exercises 101–107 use the following definitions. Let  $U$  be a universal set and let  $X \subseteq U$ . Define

$$C_X(x) = \begin{cases} 1 & \text{if } x \in X \\ 0 & \text{if } x \notin X. \end{cases}$$

We call  $C_X$  the characteristic function of  $X$  (in  $U$ ). (A look ahead at the next Problem-Solving Corner may help in understanding the following exercises.)

- 101.** Prove that  $C_{X \cap Y}(x) = C_X(x)C_Y(x)$  for all  $x \in U$ .
- 102.** Prove that  $C_{X \cup Y}(x) = C_X(x) + C_Y(x) - C_X(x)C_Y(x)$  for all  $x \in U$ .
- 103.** Prove that  $C_{\bar{X}}(x) = 1 - C_X(x)$  for all  $x \in U$ .
- 104.** Prove that  $C_{X-Y}(x) = C_X(x)[1 - C_Y(x)]$  for all  $x \in U$ .
- 105.** Prove that if  $X \subseteq Y$ , then  $C_X(x) \leq C_Y(x)$  for all  $x \in U$ .
- 106.** Find a formula for  $C_{X \Delta Y}$ . ( $X \Delta Y$  is the symmetric difference of  $X$  and  $Y$ . The definition is given before Exercise 101, Section 1.1.)
- 107.** Prove that the function  $f$  from  $\mathcal{P}(U)$  to the set of characteristic functions in  $U$  defined by

$$f(X) = C_X$$

is one-to-one and onto.

- 108.** Let  $X$  and  $Y$  be sets. Prove that there is a one-to-one function from  $X$  to  $Y$  if and only if there is a function from  $Y$  onto  $X$ .

A binary operator  $f$  on a set  $X$  is commutative if  $f(x, y) = f(y, x)$  for all  $x, y \in X$ . In Exercises 109–113, state whether the given function  $f$  is a binary operator on the set  $X$ . If  $f$  is not a binary operator, state why. State whether or not each binary operator is commutative.

- 109.**  $f(x, y) = x + y$ ,  $X = \{1, 2, \dots\}$
- 110.**  $f(x, y) = x - y$ ,  $X = \{1, 2, \dots\}$
- 111.**  $f(x, y) = x \cup y$ ,  $X = \mathcal{P}(\{1, 2, 3, 4\})$
- 112.**  $f(x, y) = x/y$ ,  $X = \{0, 1, 2, \dots\}$
- 113.**  $f(x, y) = x^2 + y^2 - xy$ ,  $X = \{1, 2, \dots\}$

In Exercises 114 and 115, give an example of a unary operator [different from  $f(x) = x$ , for all  $x$ ] on the given set.

- 114.**  $\{\dots, -2, -1, 0, 1, 2, \dots\}$
- 115.** The set of all finite subsets of  $\{1, 2, 3, \dots\}$
- 116.** Prove that if  $f$  is a one-to-one, onto function from  $X$  to  $Y$ , then

$$\{(y, x) \mid (x, y) \in f\}$$

is a one-to-one, onto function from  $Y$  to  $X$ .

In Exercises 117–119, if the statement is true for all real numbers, prove it; otherwise, give a counterexample.

- 117.**  $\lceil x + 3 \rceil = \lceil x \rceil + 3$
- 118.**  $\lceil x + y \rceil = \lceil x \rceil + \lceil y \rceil$
- 119.**  $\lfloor x + y \rfloor = \lfloor x \rfloor + \lfloor y \rfloor$
- 120.** Prove that if  $n$  is an odd integer,

$$\left\lfloor \frac{n^2}{4} \right\rfloor = \left( \frac{n-1}{2} \right) \left( \frac{n+1}{2} \right).$$

121. Prove that if  $n$  is an odd integer,

$$\left\lceil \frac{n^2}{4} \right\rceil = \frac{n^2 + 3}{4}.$$

122. Find a value for  $x$  for which  $\lceil 2x \rceil = 2\lceil x \rceil - 1$ .

123. Prove that  $2\lceil x \rceil - 1 \leq \lceil 2x \rceil \leq 2\lceil x \rceil$  for all real numbers  $x$ .

124. Prove that for all real numbers  $x$  and integers  $n$ ,  $\lceil x \rceil = n$  if and only if there exists  $\varepsilon$ ,  $0 \leq \varepsilon < 1$ , such that  $x + \varepsilon = n$ .

125. State and prove a result analogous to Exercise 124 for  $\lfloor x \rfloor$ .

The months with Friday the 13th in year  $x$  are found in row

$$y = \left( x + \left\lfloor \frac{x-1}{4} \right\rfloor - \left\lfloor \frac{x-1}{100} \right\rfloor + \left\lfloor \frac{x-1}{400} \right\rfloor \right) \bmod 7$$

in the appropriate column:

| <b>y</b> | <b>Non-Leap Year</b>            | <b>Leap Year</b>        |
|----------|---------------------------------|-------------------------|
| 0        | January,<br>October             | January,<br>April, July |
| 1        | April,<br>July                  | September,<br>December  |
| 2        | September,<br>December          | June                    |
| 3        | June                            | March,<br>November      |
| 4        | February,<br>March,<br>November | February,<br>August     |
| 5        | August                          | May                     |
| 6        | May                             | October                 |

126. Find the months with Friday the 13th in 1945.

127. Find the months with Friday the 13th in the current year.

128. Find the months with Friday the 13th in 2040.

## Problem-Solving Corner

## Functions

### Problem

Let  $U$  be a universal set and let  $X \subseteq U$ . Define

$$C_X(x) = \begin{cases} 1 & \text{if } x \in X \\ 0 & \text{if } x \notin X. \end{cases}$$

[We call  $C_X$  the *characteristic function* of  $X$  (in  $U$ )]. Assume that  $X$  and  $Y$  are arbitrary subsets of the universal set  $U$ . Prove that  $C_{X \cup Y}(x) = C_X(x) + C_Y(x)$  for all  $x \in U$  if and only if  $X \cap Y = \emptyset$ .

### Attacking the Problem

First, let's be clear what we must do. Since the statement is of the form  $p$  if and only if  $q$ , we have two tasks: (1) Prove if  $p$  then  $q$ . (2) Prove if  $q$  then  $p$ . It's a good idea to write out exactly what must be proved:

If  $C_{X \cup Y}(x) = C_X(x) + C_Y(x)$  for all  $x \in U$ ,  
then  $X \cap Y = \emptyset$ . (1)

If  $X \cap Y = \emptyset$ , then  $C_{X \cup Y}(x) = C_X(x) + C_Y(x)$   
for all  $x \in U$ . (2)

Consider the first statement in which we assume that  $C_{X \cup Y}(x) = C_X(x) + C_Y(x)$  for all  $x \in U$  and prove that  $X \cap Y = \emptyset$ . How do we prove that a set,  $X \cap Y$  in this case, is the empty set? We have to show that  $X \cap Y$  has no elements. How do we do that? There are

several possibilities, but one thing that comes to mind is another question: What if  $X \cap Y$  had an element? This suggests that we might prove the first statement by contradiction or by proving its contrapositive. If we let

$p$ :  $C_{X \cup Y}(x) = C_X(x) + C_Y(x)$  for all  $x \in U$

$q$ :  $X \cap Y = \emptyset$ ,

the contrapositive is  $\neg q \rightarrow \neg p$ . Now the negation of  $q$  is

$\neg q : X \cap Y \neq \emptyset$ ,

and, using De Morgan's law (roughly, negating  $\forall$  results in  $\exists$ ), the negation of  $p$  is

$\neg p : C_{X \cup Y}(x) \neq C_X(x) + C_Y(x)$  for at least one  $x \in U$ .

Thus, the contrapositive is

If  $X \cap Y \neq \emptyset$ , then  $C_{X \cup Y}(x) \neq C_X(x) + C_Y(x)$   
for at least one  $x \in U$ . (3)

For the second statement, we assume that  $X \cap Y = \emptyset$  and prove that  $C_{X \cup Y}(x) = C_X(x) + C_Y(x)$  for all  $x \in U$ . Presumably, we can just use the definition of  $C_X$  to compute both sides of the equation for all  $x \in U$  and verify that the two sides are equal. The definition of  $C_X$  suggests that we use proof by cases:  $x \in X \cup Y$  (when  $C_{X \cup Y}(x) = 1$ ) and  $x \notin X \cup Y$  (when  $C_{X \cup Y}(x) = 0$ ).

## Finding a Solution

We first consider proving the contrapositive (3) of statement (1). Since we assume that  $X \cap Y \neq \emptyset$ , there exists an element  $x \in X \cap Y$ . Now let's compare the values of the expressions  $C_{X \cup Y}(x)$  and  $C_X(x) + C_Y(x)$ . Since  $x \in X \cup Y$ ,  $C_{X \cup Y}(x) = 1$ . Since  $x \in X \cap Y$ ,  $x \in X$  and  $x \in Y$ . Therefore

$$C_X(x) + C_Y(x) = 1 + 1 = 2.$$

We have proved that

$$C_{X \cup Y}(x) \neq C_X(x) + C_Y(x) \text{ for at least one } x \in U.$$

Now consider proving the statement (2). This time we assume that  $X \cap Y = \emptyset$ . Let's compute each side of the equation

$$C_{X \cup Y}(x) = C_X(x) + C_Y(x) \quad (4)$$

for each  $x \in U$ . As suggested earlier, we consider the cases:  $x \in X \cup Y$  and  $x \notin X \cup Y$ . If  $x \in X \cup Y$ , then  $C_{X \cup Y}(x) = 1$ . Since  $X \cap Y = \emptyset$ , either  $x \in X$  or  $x \in Y$  but not both. Therefore,

$$C_X(x) + C_Y(x) = 1 + 0 = 1 = C_{X \cup Y}(x)$$

or

$$C_X(x) + C_Y(x) = 0 + 1 = 1 = C_{X \cup Y}(x).$$

Equation (4) is true if  $x \in X \cup Y$ .

If  $x \notin X \cup Y$ , then  $C_{X \cup Y}(x) = 0$ . But if  $x \notin X \cup Y$ , then  $x \notin X$  and  $x \notin Y$ . Therefore,

$$C_X(x) + C_Y(x) = 0 + 0 = 0 = C_{X \cup Y}(x).$$

Equation (4) is true if  $x \notin X \cup Y$ . Thus it is true for all  $x \in U$ .

## Formal Solution

The formal proof could be written as follows.

CASE →: If  $C_{X \cup Y}(x) = C_X(x) + C_Y(x)$  for all  $x \in U$ , then  $X \cap Y = \emptyset$ .

We prove the equivalent contrapositive

If  $X \cap Y \neq \emptyset$ , then  $C_{X \cup Y}(x) \neq C_X(x) + C_Y(x)$  for at least one  $x \in U$ .

Since  $X \cap Y \neq \emptyset$ , there exists  $x \in X \cap Y$ . Since  $x \in X \cup Y$ ,  $C_{X \cup Y}(x) = 1$ . Since  $x \in X \cap Y$ ,  $x \in X$  and  $x \in Y$ . Therefore

$$C_X(x) + C_Y(x) = 1 + 1 = 2.$$

Thus,

$$C_{X \cup Y}(x) \neq C_X(x) + C_Y(x).$$

CASE ←: If  $X \cap Y = \emptyset$ , then  $C_{X \cup Y}(x) = C_X(x) + C_Y(x)$  for all  $x \in U$ .

Suppose that  $x \in X \cup Y$ . Then  $C_{X \cup Y}(x) = 1$ . Since  $X \cap Y = \emptyset$ , either  $x \in X$  or  $x \in Y$  but not both. Therefore,  $C_X(x) + C_Y(x) = 1$  and (4) holds.

If  $x \notin X \cup Y$ , then  $C_{X \cup Y}(x) = 0$ . If  $x \notin X \cup Y$ , then  $x \notin X$  and  $x \notin Y$ . Therefore,  $C_X(x) + C_Y(x) = 0$ . Again, (4) holds. Therefore (4) holds for all  $x \in U$ .

## Summary of Problem-Solving Techniques

- Write out exactly what must be proved.
- Instead of proving  $p \rightarrow q$  directly, consider proving its contrapositive  $\neg q \rightarrow \neg p$  or a proof by contradiction.
- For statements involving negation, De Morgan's laws can be very helpful.
- Look for definitions and theorems relevant to the expressions mentioned in the statements to be proved.
- A definition that involves cases suggests a proof by cases.

## 3.2 Sequences and Strings

Blue Taxi Inc. charges \$1 for the first mile and 50 cents for each additional mile. The following table shows the cost of traveling from 1 to 10 miles. In general, the cost  $C_n$  of traveling  $n$  miles is 1.00 (the cost of traveling the first mile) plus 0.50 times the number  $(n - 1)$  of additional miles. That is,  $C_n = 1 + 0.5(n - 1)$ . As examples,

$$C_1 = 1 + 0.5(1 - 1) = 1 + 0.5 \cdot 0 = 1,$$

$$C_5 = 1 + 0.5(5 - 1) = 1 + 0.5 \cdot 4 = 1 + 2 = 3.$$

| Mileage | Cost   |
|---------|--------|
| 1       | \$1.00 |
| 2       | 1.50   |
| 3       | 2.00   |
| 4       | 2.50   |
| 5       | 3.00   |
| 6       | 3.50   |
| 7       | 4.00   |
| 8       | 4.50   |
| 9       | 5.00   |
| 10      | 5.50   |

The list of fares

$$C_1 = 1.00, \quad C_2 = 1.50, \quad C_3 = 2.00, \quad C_4 = 2.50, \quad C_5 = 3.00, \\ C_6 = 3.50, \quad C_7 = 4.00, \quad C_8 = 4.50, \quad C_9 = 5.00, \quad C_{10} = 5.50$$

furnishes an example of a **sequence**, which is a special type of function in which the domain consists of integers.

### Go Online

For more on sequences, see  
[goo.gl/V3y4pS](http://goo.gl/V3y4pS)

**Definition 3.2.1** ► A sequence  $s$  is a function whose domain  $D$  is a subset of integers. The notation  $s_n$  is typically used instead of the more general function notation  $s(n)$ . The term  $n$  is called the *index* of the sequence. If  $D$  is a finite set, we call  $s$  a *finite sequence*; otherwise,  $s$  is an *infinite sequence*.

For the sequence of fares for Blue Taxi, Inc., the domain of the sequence  $C$  is the subset of integers  $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ . The sequence  $C$  is a finite sequence.

A sequence  $s$  is denoted  $s$  or  $\{s_n\}$  if  $n$  is the index of the sequence. If, for example, the domain is the set of positive integers  $\mathbf{Z}^+$ ,  $s$  or  $\{s_n\}$  denotes the entire sequence  $s_1, s_2, s_3, \dots$ . We use the notation  $s_n$  to denote the single element of the sequence  $s$  at index  $n$ . In this book, we will frequently use  $\mathbf{Z}^+$  or  $\mathbf{Z}^{nonneg}$  as the domain of a sequence.

### Example 3.2.2

Consider the sequence  $s: 2, 4, 6, \dots, 2n, \dots$ . The first element of the sequence is 2, the second element of the sequence is 4, and so on. The  $n$ th element of the sequence is  $2n$ . If the domain of  $s$  is  $\mathbf{Z}^+$ , we have  $s_1 = 2, s_2 = 4, s_3 = 6, \dots, s_n = 2n, \dots$ . The sequence  $s$  is an infinite sequence.

### Example 3.2.3

Consider the sequence  $t: a, a, b, a, b$ . The first element of the sequence is  $a$ , the second element of the sequence is  $a$ , and so on. If the domain of  $t$  is  $\{1, 2, 3, 4, 5\}$ , we have  $t_1 = a, t_2 = a, t_3 = b, t_4 = a$ , and  $t_5 = b$ . The sequence  $t$  is a finite sequence.

If the domain of a sequence  $s$  is the (infinite) set of consecutive integers  $\{k, k+1, k+2, \dots\}$  and the index of  $s$  is  $n$ , we can denote the sequence  $s$  as  $\{s_n\}_{n=k}^{\infty}$ . For example, a sequence  $s$  whose domain is  $\mathbf{Z}^{nonneg}$  can be denoted  $\{s_n\}_{n=0}^{\infty}$ . If the domain of a sequence  $s$  is the set of (finite) consecutive integers  $\{i, i+1, \dots, j\}$ , we can denote the sequence  $s$  as  $\{s_n\}_{n=i}^j$ . For example, a sequence  $s$  whose domain is  $\{-1, 0, 1, 2, 3\}$  can be denoted  $\{s_n\}_{n=-1}^3$ .

### Example 3.2.4

The sequence  $\{u_n\}$  defined by the rule  $u_n = n^2 - 1$ , for all  $n \geq 0$ , can be denoted  $\{u_n\}_{n=0}^{\infty}$ . The name of the index can be chosen in any convenient way. For example, the sequence  $u$  can also be denoted  $\{u_m\}_{m=0}^{\infty}$ . The formula for the term having index  $m$  is  $u_m = m^2 - 1$ , for all  $m \geq 0$ .

**Example 3.2.5**

Define a sequence  $b$  by the rule  $b_n$  is the  $n$ th letter in the word *digital*. If the domain of  $b$  is  $\{1, 2, \dots, 7\}$ , then  $b_1 = d$ ,  $b_2 = b_4 = i$ , and  $b_7 = l$ .

**Example 3.2.6**

If  $x$  is the sequence defined by

$$x_n = \frac{1}{2^n} \quad -1 \leq n \leq 4,$$

the elements of  $x$  are  $2, 1, 1/2, 1/4, 1/8, 1/16$ .

**Example 3.2.7**

Define a sequence  $s$  as

$$s_n = 2^n + 4 \cdot 3^n \quad n \geq 0. \quad (3.2.1)$$

- (a) Find  $s_0$ .
- (b) Find  $s_1$ .
- (c) Find a formula for  $s_i$ .
- (d) Find a formula for  $s_{n-1}$ .
- (e) Find a formula for  $s_{n-2}$ .
- (f) Prove that  $\{s_n\}$  satisfies

$$s_n = 5s_{n-1} - 6s_{n-2} \quad \text{for all } n \geq 2. \quad (3.2.2)$$

**SOLUTION**

- (a) Replacing  $n$  by 0 in Definition 3.2.1, we obtain

$$s_0 = 2^0 + 4 \cdot 3^0 = 5.$$

- (b) Replacing  $n$  by 1 in Definition 3.2.1, we obtain

$$s_1 = 2^1 + 4 \cdot 3^1 = 14.$$

- (c) Replacing  $n$  by  $i$  in Definition 3.2.1, we obtain

$$s_i = 2^i + 4 \cdot 3^i.$$

- (d) Replacing  $n$  by  $n - 1$  in Definition 3.2.1, we obtain

$$s_{n-1} = 2^{n-1} + 4 \cdot 3^{n-1}.$$

- (e) Replacing  $n$  by  $n - 2$  in Definition 3.2.1, we obtain

$$s_{n-2} = 2^{n-2} + 4 \cdot 3^{n-2}.$$

- (f) To prove equation (3.2.2), we will replace  $s_{n-1}$  and  $s_{n-2}$  in the right side of equation (3.2.2) by the formulas of parts (d) and (e). We will then use algebra to show that the result is equal to  $s_n$ . We obtain

$$\begin{aligned} 5s_{n-1} - 6s_{n-2} &= 5(2^{n-1} + 4 \cdot 3^{n-1}) - 6(2^{n-2} + 4 \cdot 3^{n-2}) \\ &= (5 \cdot 2 - 6)2^{n-2} + (5 \cdot 4 \cdot 3 - 6 \cdot 4)3^{n-2} \\ &= 4 \cdot 2^{n-2} + 36 \cdot 3^{n-2} \\ &= 2^2 2^{n-2} + (4 \cdot 3^2)3^{n-2} \\ &= 2^n + 4 \cdot 3^n = s_n. \end{aligned}$$

The techniques shown in the example will be useful in checking solutions of recurrence relations in Chapter 7.

Two important types of sequences are increasing sequences and decreasing sequences and their relatives, nonincreasing sequences and nondecreasing sequences. A sequence  $s$  is **increasing** if for all  $i$  and  $j$  in the domain of  $s$ , if  $i < j$ , then  $s_i < s_j$ . A sequence  $s$  is **decreasing** if for all  $i$  and  $j$  in the domain of  $s$ , if  $i < j$ , then  $s_i > s_j$ . A sequence  $s$  is **nondecreasing** if for all  $i$  and  $j$  in the domain of  $s$ , if  $i < j$ , then  $s_i \leq s_j$ . (A nondecreasing sequence is like an increasing sequence except that, in the condition on  $s$ , " $<$ " is replaced by " $\leq$ ".) A sequence  $s$  is **nonincreasing** if for all  $i$  and  $j$  in the domain of  $s$ , if  $i < j$ , then  $s_i \geq s_j$ . (A nonincreasing sequence is like a decreasing sequence except that, in the condition on  $s$ , " $>$ " is replaced by " $\geq$ ".) Notice that if the domain of a sequence  $s$  is a set of consecutive integers, if  $s_i < s_{i+1}$  for all  $i$  for which  $i$  and  $i+1$  are in the domain of  $s$ , then  $s$  is increasing. Similar remarks apply to decreasing, nondecreasing, and nonincreasing sequences. (We leave formal justification for this last comment to Exercise 4.)

### **Example 3.2.8**

The sequence 2, 5, 13, 104, 300 is increasing and nondecreasing.

### **Example 3.2.9**

The sequence

$$a_i = \frac{1}{i} \quad i \geq 1,$$

is decreasing and nonincreasing.

### **Example 3.2.10**

The sequence 100, 90, 90, 74, 74, 74, 30 is nonincreasing, but it is *not* decreasing.

### **Example 3.2.11**

The sequence 100 (consisting of a single element) is increasing, decreasing, nonincreasing, and nondecreasing since there are no distinct values of  $i$  and  $j$  for which both  $i$  and  $j$  are indexes.

One way to form a new sequence from a given sequence is to retain only certain terms of the original sequence, maintaining the order of terms in the given sequence. The resulting sequence is called a **subsequence** of the original sequence.

**Definition 3.2.12** ► Let  $s$  be a sequence. A *subsequence* of  $s$  is a sequence obtained from  $s$  by choosing certain terms of  $s$  in the same order in which they appear in  $s$ .

### **Example 3.2.13**

The sequence

$$b, c \tag{3.2.3}$$

is a subsequence of the sequence

$$a, a, b, c, q. \tag{3.2.4}$$

Subsequence (3.2.3) is obtained by choosing the third and fourth terms from sequence (3.2.4).

Notice that the sequence  $c, b$  is *not* a subsequence of sequence (3.2.4) since the order of terms in the sequence (3.2.4) is not maintained.

**Example 3.2.14** The sequence

$$2, 4, 8, 16, \dots, 2^k, \dots \quad (3.2.5)$$

is a subsequence of the sequence

$$2, 4, 6, 8, 10, 12, 14, 16, \dots, 2n, \dots \quad (3.2.6)$$

Subsequence (3.2.5) is obtained by choosing the first, second, fourth, eighth, and so on, terms from sequence (3.2.6). 

We turn to the notation for a subsequence. A subsequence of a sequence  $s$  is obtained by choosing certain terms from  $s$ . We let  $n_1, n_2, n_3, \dots$  denote the indexes (in order) of  $s$  of the terms that are selected to obtain the subsequence. Thus the subsequence can be denoted  $\{s_{n_k}\}$ .

**Example 3.2.15** Let us return to the sequence (3.2.4) in Example 3.2.13, which we now call  $t$ :

$$t_1 = a, \quad t_2 = a, \quad t_3 = b, \quad t_4 = c, \quad t_5 = q.$$

The subsequence  $b, c$  is obtained from  $t$  by selecting the third and fourth terms of  $t$ . Thus  $n_1 = 3, n_2 = 4$ , and the subsequence  $b, c$  is  $t_3, t_4$ , or  $t_{n_1}, t_{n_2}$ . 

**Example 3.2.16** Let us return to the sequence (3.2.6) in Example 3.2.14. We let  $s_n = 2n$ , so the sequence (3.2.6) can be denoted  $\{s_n\}_{n=1}^{\infty}$ . The subsequence (3.2.5) is obtained from  $s$  by selecting the first, second, fourth, eighth, and so on, terms from  $s$ ; thus

$$n_1 = 1, \quad n_2 = 2, \quad n_3 = 4, \quad n_4 = 8, \dots$$

We see that  $n_k = 2^{k-1}, k = 1, 2, \dots$ . The subsequence (3.2.5)  $\{s_{n_k}\}_{k=1}^{\infty}$  is defined by

$$s_{n_k} = s_{2^{k-1}} = 2 \cdot 2^{k-1} = 2^k.$$

In the notation for the original sequence  $\{s_n\}_{n=1}^{\infty}$ ,  $n$  is the index of the sequence  $s$ . In the notation for the subsequence  $\{s_{n_k}\}_{k=1}^{\infty}$ ,  $n$  is used in a totally different way;  $n$  is itself a *sequence*, namely, the sequence of indexes selected to form the subsequence. Furthermore,  $k$  is the index of the sequence  $n$ . It would have been nice to avoid having  $n$  used in two different ways, but tradition demands that we continue this abuse of notation. 

Two important operations on numerical sequences are adding and multiplying terms.

**Definition 3.2.17** ► If  $\{a_i\}_{i=m}^n$  is a sequence, we define

$$\sum_{i=m}^n a_i = a_m + a_{m+1} + \dots + a_n, \quad \prod_{i=m}^n a_i = a_m \cdot a_{m+1} \cdots a_n.$$

**Go Online**

For more on the sigma notation, see  
[goo.gl/V3y4pS](http://goo.gl/V3y4pS)

The formalism

$$\sum_{i=m}^n a_i \quad (3.2.7)$$

is called the *sum* (or *sigma*) *notation* and

$$\prod_{i=m}^n a_i \quad (3.2.8)$$

is called the *product notation*.

In (3.2.7) or (3.2.8),  $i$  is called the *index*,  $m$  is called the *lower limit*, and  $n$  is called the *upper limit*. 

**Example 3.2.18** Let  $a$  be the sequence defined by  $a_n = 2n$ ,  $n \geq 1$ . Then

$$\begin{aligned}\sum_{i=1}^3 a_i &= a_1 + a_2 + a_3 = 2 + 4 + 6 = 12, \\ \prod_{i=1}^3 a_i &= a_1 \cdot a_2 \cdot a_3 = 2 \cdot 4 \cdot 6 = 48.\end{aligned}$$


**Example 3.2.19**

The geometric sum  $a + ar + ar^2 + \cdots + ar^n$  can be rewritten compactly using the sum notation as  $\sum_{i=0}^n ar^i$ . 

It is sometimes useful to change not only the name of the index, but to change its limits as well. (The process is analogous to changing the variable in an integral in calculus.)

**Example 3.2.20**

**Changing the Index and Limits in a Sum** Rewrite the sum  $\sum_{i=0}^n ir^{n-i}$ , replacing the index  $i$  by  $j$ , where  $i = j - 1$ .

**SOLUTION** Since  $i = j - 1$ , the term  $ir^{n-i}$  becomes

$$(j-1)r^{n-(j-1)} = (j-1)r^{n-j+1}.$$

Since  $j = i + 1$ , when  $i = 0$ ,  $j = 1$ . Thus the lower limit for  $j$  is 1. Similarly, when  $i = n$ ,  $j = n + 1$ , and the upper limit for  $j$  is  $n + 1$ . Therefore,

$$\sum_{i=0}^n ir^{n-i} = \sum_{j=1}^{n+1} (j-1)r^{n-j+1}.$$


**Example 3.2.21**

Let  $a$  be the sequence defined by the rule  $a_i = 2(-1)^i$ ,  $i \geq 0$ . Find a formula for the sequence  $s$  defined by  $s_n = \sum_{i=0}^n a_i$ .

**SOLUTION** We find that

$$\begin{aligned}s_n &= 2(-1)^0 + 2(-1)^1 + 2(-1)^2 + \cdots + 2(-1)^n \\ &= 2 - 2 + 2 - \cdots \pm 2 = \begin{cases} 2 & \text{if } n \text{ is even} \\ 0 & \text{if } n \text{ is odd.} \end{cases}\end{aligned}$$


Sometimes the sum and product notations are modified to denote sums and products indexed over arbitrary sets of integers. Formally, if  $S$  is a finite set of integers and  $a$  is a sequence,  $\sum_{i \in S} a_i$  denotes the sum of the elements  $\{a_i \mid i \in S\}$ . Similarly,  $\prod_{i \in S} a_i$  denotes the product of the elements  $\{a_i \mid i \in S\}$ .

**Example 3.2.22** If  $S$  denotes the set of prime numbers less than 20,

$$\sum_{i \in S} \frac{1}{i} = \frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \frac{1}{11} + \frac{1}{13} + \frac{1}{17} + \frac{1}{19} = 1.455.$$

A *string* is a finite sequence of characters. In programming languages, strings can be used to denote text. For example, in Java

"Let's read Rolling Stone."

denotes the string consisting of the sequence of characters

Let's read Rolling Stone.

(The double quotes " mark the start and end of the string.)

Within a computer, *bit strings* (strings of 0's and 1's) represent data and instructions to execute. As we will see in Section 5.2, the bit string 101111 represents the number 47.

**Definition 3.2.23** ► A *string over  $X$* , where  $X$  is a finite set, is a finite sequence of elements from  $X$ .

**Example 3.2.24** Let  $X = \{a, b, c\}$ . If we let

$$\beta_1 = b, \quad \beta_2 = a, \quad \beta_3 = a, \quad \beta_4 = c,$$

we obtain a string over  $X$ . This string is written  $baac$ .

Since a string is a sequence, order is taken into account. For example, the string  $baac$  is different from the string  $acab$ .

Repetitions in a string can be specified by superscripts. For example, the string  $bbaaac$  may be written  $b^2a^3c$ .

The string with no elements is called the **null string** and is denoted  $\lambda$ . We let  $X^*$  denote the set of all strings over  $X$ , including the null string, and we let  $X^+$  denote the set of all nonnull strings over  $X$ .

**Example 3.2.25** Let  $X = \{a, b\}$ . Some elements in  $X^*$  are  $\lambda$ ,  $a$ ,  $b$ ,  $abab$ , and  $b^{20}a^5ba$ .

The **length** of a string  $\alpha$  is the number of elements in  $\alpha$ . The length of  $\alpha$  is denoted  $|\alpha|$ .

**Example 3.2.26** If  $\alpha = aabab$  and  $\beta = a^3b^4a^{32}$ , then  $|\alpha| = 5$  and  $|\beta| = 39$ .

If  $\alpha$  and  $\beta$  are two strings, the string consisting of  $\alpha$  followed by  $\beta$ , written  $\alpha\beta$ , is called the **concatenation** of  $\alpha$  and  $\beta$ .

**Example 3.2.27** If  $\gamma = aab$  and  $\theta = cabd$ , then

$$\gamma\theta = aabcabd, \quad \theta\gamma = cabdaab, \quad \gamma\lambda = \gamma = aab, \quad \lambda\gamma = \gamma = aab.$$

**Example 3.2.28** Let  $X = \{a, b, c\}$ . If we define  $f(\alpha, \beta) = \alpha\beta$ , where  $\alpha$  and  $\beta$  are strings over  $X$ , then  $f$  is a binary operator on  $X^*$ .

A **substring** of a string  $\alpha$  is obtained by selecting some or all consecutive elements of  $\alpha$ . The formal definition follows.

**Definition 3.2.29 ▶** A string  $\beta$  is a *substring* of the string  $\alpha$  if there are strings  $\gamma$  and  $\delta$  with  $\alpha = \gamma\beta\delta$ .

**Example 3.2.30**

The string  $\beta = add$  is a substring of the string  $\alpha = aaaddad$  since, if we take  $\gamma = aa$  and  $\delta = ad$ , we have  $\alpha = \gamma\beta\delta$ . Note that if  $\beta$  is a substring of  $\alpha$ ,  $\gamma$  is the part of  $\alpha$  that precedes  $\beta$  (in  $\alpha$ ), and  $\delta$  is the part of  $\alpha$  that follows  $\beta$  (in  $\alpha$ ).

**Example 3.2.31**

Let  $X = \{a, b\}$ . If  $\alpha \in X^*$ , let  $\alpha^R$  denote  $\alpha$  written in reverse. For example, if  $\alpha = abb$ ,  $\alpha^R = bba$ . Define a function from  $X^*$  to  $X^*$  as  $f(\alpha) = \alpha^R$ . Prove that  $f$  is a bijection.

**SOLUTION** We must show that  $f$  is one-to-one and onto  $X^*$ . We first show that  $f$  is one-to-one. We must show that if  $f(\alpha) = f(\beta)$ , then  $\alpha = \beta$ . So suppose that  $f(\alpha) = f(\beta)$ . Using the definition of  $f$ , we have  $\alpha^R = \beta^R$ . Reversing each side, we find that  $\alpha = \beta$ . Therefore,  $f$  is one-to-one.

Next we show that  $f$  is onto  $X^*$ . We must show that if  $\beta \in X^*$ , there exists  $\alpha \in X^*$  such that  $f(\alpha) = \beta$ . So suppose that  $\beta \in X^*$ . If we let  $\alpha = \beta^R$ , we have

$$f(\alpha) = \alpha^R = (\beta^R)^R = \beta$$

since if we twice reverse a string, we obtain the original string. Therefore,  $f$  is onto  $X^*$ . We have proved that  $f$  is a bijection.

**Example 3.2.32**

Let  $X = \{a, b\}$ . Define a function from  $X^* \times X^*$  to  $X^*$  as  $f(\alpha, \beta) = \alpha\beta$ . Is  $f$  one-to-one? Is  $f$  onto  $X^*$ ?

**SOLUTION** We try to prove that  $f$  is one-to-one. If we succeed, this part of the example is complete. If we fail, we may learn how to construct a counterexample. So suppose that  $f(\alpha_1, \beta_1) = f(\alpha_2, \beta_2)$ . We have to prove that  $\alpha_1 = \beta_1$  and  $\alpha_2 = \beta_2$ . Using the definition of  $f$ , we have  $\alpha_1\beta_1 = \alpha_2\beta_2$ . Can we conclude that  $\alpha_1 = \beta_1$  and  $\alpha_2 = \beta_2$ ? No! It is possible to concatenate different strings and produce the same string. For example,  $baa = \alpha_1\beta_1$  if we set  $\alpha_1 = b$  and  $\beta_1 = aa$ . Also,  $baa = \alpha_2\beta_2$  if we set  $\alpha_2 = ba$  and  $\beta_2 = a$ . Therefore  $f$  is not one-to-one. We could write up this part of the solution as follows.

If we set  $\alpha_1 = b$ ,  $\beta_1 = aa$ ,  $\alpha_2 = ba$ , and  $\beta_2 = a$ , then  $f(\alpha_1, \beta_1) = baa = f(\alpha_2, \beta_2)$ . Since  $\alpha_1 \neq \alpha_2$ ,  $f$  is not one-to-one.

The function  $f$  is onto  $X^*$  if given any string  $\gamma \in X^*$ , there exist  $(\alpha, \beta) \in X^* \times X^*$  such that  $f(\alpha, \beta) = \gamma$ . In words,  $f$  is onto  $X^*$  if every string in  $X^*$  is the concatenation of two strings, each in  $X^*$ . Since concatenating a string  $\alpha$  with the null string  $\lambda$  does not change  $\alpha$ , every string in  $X^*$  is the concatenation of two strings, each in  $X^*$ . This part of the solution could be written up as follows.

Let  $\alpha \in X^*$ . Then  $f(\alpha, \lambda) = \alpha\lambda = \alpha$ . Therefore  $f$  is onto  $X^*$ .

### 3.2 Problem-Solving Tips

A sequence is a special type of function; the domain is a set of integers. If  $a_1, a_2, \dots$  is a sequence, the numbers  $1, 2, \dots$  are called indexes. Index 1 identifies the first element of the sequence  $a_1$ ; index 2 identifies the second element of the sequence  $a_2$ ; and so on.

In this book, “increasing sequence” means *strictly* increasing; that is, the sequence  $a$  is increasing if for all  $i$  and  $j$  in the domain of  $a$ , if  $i < j$ , then  $a_i < a_j$ . We require that  $a_i$  is strictly less than (*not* less than or equal to)  $a_j$ . Allowing equality yields what we call in this book a “nondecreasing sequence.” That is, the sequence  $a$  is nondecreasing if for all  $i$  and  $j$  in the domain of  $a$ , if  $i < j$ , then  $a_i \leq a_j$ . Similar remarks apply to decreasing sequences and nonincreasing sequences.

## 3.2 Review Exercises

1. Define *sequence*.
2. What is an index in a sequence?
3. Define *increasing sequence*.
4. Define *decreasing sequence*.
5. Define *nonincreasing sequence*.
6. Define *nondecreasing sequence*.
7. Define *subsequence*.
8. What is  $\sum_{i=m}^n a_i$ ?
9. What is  $\prod_{i=m}^n a_i$ ?
10. Define *string*.
11. Define *null string*.
12. If  $X$  is a finite set, what is  $X^*$ ?
13. If  $X$  is a finite set, what is  $X^+$ ?
14. Define *length of a string*. How is the length of the string  $\alpha$  denoted?
15. Define *concatenation of strings*. How is the concatenation of strings  $\alpha$  and  $\beta$  denoted?
16. Define *substring*.

## 3.2 Exercises

*Answer 1–3 for the sequence  $\{s_n\}_{n=1}^6$  defined by*  
 $c, d, d, c, d, c.$

1. Find  $s_1$ .
2. Find  $s_4$ .
3. Write  $s$  as a string.
4. Let  $s$  be a sequence whose domain  $D$  is a set of consecutive integers. Prove that if  $s_i < s_{i+1}$  for all  $i$  for which  $i$  and  $i+1$  are in  $D$ , then  $s$  is increasing. Hint: Let  $i \in D$ . Use induction on  $j$  to show that  $s_i < s_j$  for all  $j$ ,  $i < j$  and  $j \in D$ .

*In Exercises 5–9, tell whether the sequence  $s$  defined by  $s_n = 2^n - n^2$  is*

- (a) *increasing*
- (b) *decreasing*
- (c) *nonincreasing*
- (d) *nondecreasing*

*for the given domain  $D$ .*

5.  $D = \{0, 1\}$
6.  $D = \{0, 1, 2, 3\}$
7.  $D = \{1, 2, 3\}$
8.  $D = \{1, 2, 3, 4\}$
9.  $D = \{n \mid n \in \mathbb{Z}, n \geq 3\}$

*Answer 10–22 for the sequence  $t$  defined by*

$$t_n = 2n - 1, \quad n \geq 1.$$

10. Find  $t_3$ .
11. Find  $t_7$ .
12. Find  $t_{100}$ .
13. Find  $t_{2077}$ .
14. Find  $\sum_{i=1}^3 t_i$ .
15. Find  $\sum_{i=3}^7 t_i$ .
16. Find  $\prod_{i=1}^3 t_i$ .
17. Find  $\prod_{i=3}^6 t_i$ .

**18.** Find a formula that represents this sequence as a sequence whose lower index is 0.

19. Is  $t$  increasing?
20. Is  $t$  decreasing?
21. Is  $t$  nonincreasing?
22. Is  $t$  nondecreasing?

*Answer 23–30 for the sequence  $v$  defined by*

$$v_n = n! + 2, \quad n \geq 1.$$

23. Find  $v_3$ .
24. Find  $v_4$ .
25. Find  $\sum_{i=1}^4 v_i$ .
26. Find  $\sum_{i=3}^3 v_i$ .
27. Is  $v$  increasing?
28. Is  $v$  decreasing?
29. Is  $v$  nonincreasing?
30. Is  $v$  nondecreasing?

*Answer 31–36 for the sequence*

$$q_1 = 8, \quad q_2 = 12, \quad q_3 = 12, \quad q_4 = 28, \quad q_5 = 33.$$

31. Find  $\sum_{i=2}^4 q_i$ .
32. Find  $\sum_{k=2}^4 q_k$ .
33. Is  $q$  increasing?
34. Is  $q$  decreasing?
35. Is  $q$  nonincreasing?
36. Is  $q$  nondecreasing?

*Answer 37–40 for the sequence*

$$\tau_0 = 5, \quad \tau_2 = 5.$$

37. Is  $\tau$  increasing?
38. Is  $\tau$  decreasing?
39. Is  $\tau$  nonincreasing?
40. Is  $\tau$  nondecreasing?

*Answer 41–44 for the sequence*

$$\Upsilon_2 = 5.$$

41. Is  $\Upsilon$  increasing?
42. Is  $\Upsilon$  decreasing?

43. Is  $\Upsilon$  nonincreasing?      44. Is  $\Upsilon$  nondecreasing?

Answer 45–56 for the sequence  $a$  defined by

$$a_n = n^2 - 3n + 3, \quad n \geq 1.$$

45. Find  $\sum_{i=1}^4 a_i.$       46. Find  $\sum_{j=3}^5 a_j.$   
 47. Find  $\sum_{i=4}^4 a_i.$       48. Find  $\sum_{k=1}^6 a_k.$   
 49. Find  $\prod_{i=1}^2 a_i.$       50. Find  $\prod_{i=1}^3 a_i.$   
 51. Find  $\prod_{n=2}^3 a_n.$       52. Find  $\prod_{x=3}^4 a_x.$   
 53. Is  $a$  increasing?      54. Is  $a$  decreasing?  
 55. Is  $a$  nonincreasing?      56. Is  $a$  nondecreasing?

Answer 57–64 for the sequence  $b$  defined by  $b_n = n(-1)^n, n \geq 1.$

57. Find  $\sum_{i=1}^4 b_i.$       58. Find  $\sum_{i=1}^{10} b_i.$

59. Find a formula for the sequence  $c$  defined by

$$c_n = \sum_{i=1}^n b_i.$$

60. Find a formula for the sequence  $d$  defined by

$$d_n = \prod_{i=1}^n b_i.$$

61. Is  $b$  increasing?      62. Is  $b$  decreasing?  
 63. Is  $b$  nonincreasing?      64. Is  $b$  nondecreasing?

Answer 65–72 for the sequence  $\Omega$  defined by  $\Omega_n = 3$  for all  $n.$

65. Find  $\sum_{i=1}^3 \Omega_i.$       66. Find  $\sum_{i=1}^{10} \Omega_i.$

67. Find a formula for the sequence  $c$  defined by

$$c_n = \sum_{i=1}^n \Omega_i.$$

68. Find a formula for the sequence  $d$  defined by

$$d_n = \prod_{i=1}^n \Omega_i.$$

69. Is  $\Omega$  increasing?      70. Is  $\Omega$  decreasing?  
 71. Is  $\Omega$  nonincreasing?      72. Is  $\Omega$  nondecreasing?

Answer 73–79 for the sequence  $x$  defined by

$$x_1 = 2, \quad x_n = 3 + x_{n-1}, \quad n \geq 2.$$

73. Find  $\sum_{i=1}^3 x_i.$       74. Find  $\sum_{i=1}^{10} x_i.$

75. Find a formula for the sequence  $c$  defined by

$$c_n = \sum_{i=1}^n x_i.$$

76. Is  $x$  increasing?      77. Is  $x$  decreasing?  
 78. Is  $x$  nonincreasing?      79. Is  $x$  nondecreasing?

Answer 80–87 for the sequence  $w$  defined by

$$w_n = \frac{1}{n} - \frac{1}{n+1}, \quad n \geq 1.$$

80. Find  $\sum_{i=1}^3 w_i.$       81. Find  $\sum_{i=1}^{10} w_i.$

82. Find a formula for the sequence  $c$  defined by

$$c_n = \sum_{i=1}^n w_i.$$

83. Find a formula for the sequence  $d$  defined by

$$d_n = \prod_{i=1}^n w_i.$$

84. Is  $w$  increasing?      85. Is  $w$  decreasing?  
 86. Is  $w$  nonincreasing?      87. Is  $w$  nondecreasing?

Answer 88–100 for the sequence  $a$  defined by

$$a_n = \frac{n-1}{n^2(n-2)^2}, \quad n \geq 3$$

and the sequence  $z$  defined by  $z_n = \sum_{i=3}^n a_i.$

88. Find  $a_3.$   
 89. Find  $a_4.$   
 90. Find  $z_3.$   
 91. Find  $z_4.$   
 92. Find  $z_{100}.$  Hint: Show that

$$a_n = \frac{1}{4} \left[ \frac{1}{(n-2)^2} - \frac{1}{n^2} \right]$$

and use this form in the sum. Write out  $a_3 + a_4 + a_5 + a_6$  to see what is going on.

93. Is  $a$  increasing?  
 94. Is  $a$  decreasing?  
 95. Is  $a$  nonincreasing?  
 96. Is  $a$  nondecreasing?  
 97. Is  $z$  increasing?  
 98. Is  $z$  decreasing?  
 99. Is  $z$  nonincreasing?  
 100. Is  $z$  nondecreasing?

Let  $X$  be the set of positive integers that are not perfect squares. (A perfect square  $m$  is an integer of the form  $m = i^2$  where  $i$  is an integer.) Exercises 101–107 concern the sequence  $s$  from  $X$  to  $\mathbf{Z}$  defined as follows. If  $n \in X$ , let  $s_n$  be the least integer  $a_k$  for

which there exist integers  $a_1, \dots, a_k$  with  $n < a_1 < a_2 < \dots < a_k$  such that  $n \cdot a_1 \cdots a_k$  is a perfect square. As an example, consider  $s_2$ . None of  $2 \cdot 3, 2 \cdot 4, 2 \cdot 3 \cdot 4$  is a perfect square, so  $s_2 \neq 3$  and  $s_2 \neq 4$ . If  $s_2 = 5$ ,  $2 \cdot a_1 \cdots a_k \cdot 5$  would be a perfect square for some  $a_1, \dots, a_k$ ,  $2 < a_1 < a_2 < \dots < a_k < 5$ . But then one of the  $a_i$  would be a multiple of 5, which is impossible. Therefore,  $s_2 \neq 5$ . However,  $2 \cdot 3 \cdot 6$  is a perfect square, so  $s_2 = 6$ .

101. Show that  $s_n$  is defined for every  $n \in X$ , that is, that for any  $n \in X$ , there exist integers  $a_1, \dots, a_k$  with  $n < a_1 < a_2 < \dots < a_k$  such that  $n \cdot a_1 \cdots a_k$  is a perfect square.
102. Show that  $s_n \leq 4n$  for all  $n \in X$ .
103. Find  $s_3$ .
104. Find  $s_5$ .
105. Find  $s_6$ .
- ★106. Prove that if  $p$  is a prime,  $s_p = 2p$  for all  $p \geq 5$ .
107. Prove that  $s$  is not increasing.
108. Let  $u$  be the sequence defined by

$$u_1 = 3, \quad u_n = 3 + u_{n-1}, \quad n \geq 2.$$

Find a formula for the sequence  $d$  defined by

$$d_n = \prod_{i=1}^n u_i.$$

*Exercises 109–112 refer to the sequence  $\{s_n\}$  defined by the rule*

$$s_n = 2n - 1, \quad n \geq 1.$$

109. List the first seven terms of  $s$ .

*Answer 110–112 for the subsequence of  $s$  obtained by taking the first, third, fifth, ... terms.*

110. List the first seven terms of the subsequence.
111. Find a formula for the expression  $n_k$  as described before Example 3.2.15.
112. Find a formula for the  $k$ th term of the subsequence.

*Exercises 113–116 refer to the sequence  $\{t_n\}$  defined by the rule*

$$t_n = 2^n, \quad n \geq 1.$$

113. List the first seven terms of  $t$ .

*Answer 114–116 for the subsequence of  $t$  obtained by taking the first, second, fourth, seventh, eleventh, ... terms.*

114. List the first seven terms of the subsequence.
115. Find a formula for the expression  $n_k$  as described before Example 3.2.15.
116. Find a formula for the  $k$ th term of the subsequence.

*Answer 117–120 using the sequences  $y$  and  $z$  defined by*

$$y_n = 2^n - 1, \quad z_n = n(n - 1).$$

117. Find  $\left( \sum_{i=1}^3 y_i \right) \left( \sum_{i=1}^3 z_i \right)$ . 118. Find  $\left( \sum_{i=1}^5 y_i \right) \left( \sum_{i=1}^4 z_i \right)$ .

119. Find  $\sum_{i=1}^3 y_i z_i$ . 120. Find  $\left( \sum_{i=3}^4 y_i \right) \left( \prod_{i=2}^4 z_i \right)$ .

*Answer 121–128 for the sequence  $r$  defined by*

$$r_n = 3 \cdot 2^n - 4 \cdot 5^n, \quad n \geq 0.$$

121. Find  $r_0$ .
122. Find  $r_1$ .
123. Find  $r_2$ .
124. Find  $r_3$ .
125. Find a formula for  $r_p$ .
126. Find a formula for  $r_{n-1}$ .
127. Find a formula for  $r_{n-2}$ .
128. Prove that  $\{r_n\}$  satisfies

$$r_n = 7r_{n-1} - 10r_{n-2}, \quad n \geq 2.$$

*Answer 129–136 for the sequence  $z$  defined by*

$$z_n = (2 + n)3^n, \quad n \geq 0.$$

129. Find  $z_0$ .
130. Find  $z_1$ .
131. Find  $z_2$ .
132. Find  $z_3$ .
133. Find a formula for  $z_i$ .
134. Find a formula for  $z_{n-1}$ .
135. Find a formula for  $z_{n-2}$ .
136. Prove that  $\{z_n\}$  satisfies

$$z_n = 6z_{n-1} - 9z_{n-2}, \quad n \geq 2.$$

137. Find  $b_n$ ,  $n = 1, \dots, 6$ , where

$$b_n = n + (n - 1)(n - 2)(n - 3)(n - 4)(n - 5).$$

138. Rewrite the sum

$$\sum_{i=1}^n i^2 r^{n-i},$$

replacing the index  $i$  by  $k$ , where  $i = k + 1$ .

139. Rewrite the sum

$$\sum_{k=1}^n C_{k-1} C_{n-k},$$

replacing the index  $k$  by  $i$ , where  $k = i + 1$ .

140. Let  $a$  and  $b$  be sequences, and let

$$s_k = \sum_{i=1}^k a_i.$$

Prove that

$$\sum_{k=1}^n a_k b_k = \sum_{k=1}^n s_k (b_k - b_{k+1}) + s_n b_{n+1}.$$

This equation, known as the *summation-by-parts formula*, is the discrete analog of the integration-by-parts formula in calculus.

141. Sometimes we generalize the notion of sequence as defined in this section by allowing more general indexing. Suppose

that  $\{a_{ij}\}$  is a sequence indexed over pairs of positive integers. Prove that

$$\sum_{i=1}^n \left( \sum_{j=i}^n a_{ij} \right) = \sum_{j=1}^n \left( \sum_{i=1}^j a_{ij} \right).$$

**142.** Compute the given quantity using the strings

$$\alpha = baab, \quad \beta = caaba, \quad \gamma = bbab.$$

- |                      |                         |                              |
|----------------------|-------------------------|------------------------------|
| (a) $\alpha\beta$    | (b) $\beta\alpha$       | (c) $\alpha\alpha$           |
| (d) $\beta\beta$     | (e) $ \alpha\beta $     | (f) $ \beta\alpha $          |
| (g) $ \alpha\alpha $ | (h) $ \beta\beta $      | (i) $\alpha\lambda$          |
| (j) $\lambda\beta$   | (k) $\alpha\beta\gamma$ | (l) $\beta\beta\gamma\alpha$ |

**143.** List all strings over  $X = \{0, 1\}$  of length 2.

**144.** List all strings over  $X = \{0, 1\}$  of length 2 or less.

**145.** List all strings over  $X = \{0, 1\}$  of length 3.

**146.** List all strings over  $X = \{0, 1\}$  of length 3 or less.

**147.** Find all substrings of the string  $babc$ .

**148.** Find all substrings of the string  $aabaabb$ .

**149.** Use induction to prove that

$$\sum \frac{1}{n_1 \cdot n_2 \cdots n_k} = n,$$

for all  $n \geq 1$ , where the sum is taken over all nonempty subsets  $\{n_1, n_2, \dots, n_k\}$  of  $\{1, 2, \dots, n\}$ .

**150.** Suppose that the sequence  $\{a_n\}$  satisfies  $a_1 = 0, a_2 = 1$ , and

$$a_n = (n-1)(a_{n-1} + a_{n-2}) \quad \text{for all } n \geq 3.$$

Use induction to prove that

$$\frac{a_n}{n!} = \sum_{k=0}^n \frac{(-1)^k}{k!} \quad \text{for all } n \geq 1.$$

In Exercises 151–153,  $x_1, x_2, \dots, x_n, n \geq 2$ , are real numbers satisfying  $x_1 < x_2 < \dots < x_n$ , and  $x$  is an arbitrary real number.

**151.** Prove that if  $x_1 \leq x \leq x_n$ , then

$$\sum_{i=1}^n |x - x_i| = \sum_{i=2}^{n-1} |x - x_i| + (x_n - x_1),$$

for all  $n \geq 3$ .

**152.** Prove that if  $x < x_1$  or  $x > x_n$ , then

$$\sum_{i=1}^n |x - x_i| > \sum_{i=2}^{n-1} |x - x_i| + (x_n - x_1),$$

for all  $n \geq 3$ .

**153.** A *median* of  $x_1, \dots, x_n$  is the middle value of  $x_1, \dots, x_n$  when  $n$  is odd, and any value between the two middle values of  $x_1, \dots, x_n$  when  $n$  is even. For example, if  $x_1 < x_2 < \dots < x_5$ , the median is  $x_3$ . If  $x_1 < x_2 < x_3 < x_4$ , a median is any value between  $x_2$  and  $x_3$ , including  $x_2$  and  $x_3$ .

Use Exercises 151 and 152 and mathematical induction to prove that the sum

$$\sum_{i=1}^n |x - x_i|, \tag{3.2.9}$$

$n \geq 1$ , is minimized when  $x$  is equal to a median of  $x_1, \dots, x_n$ .

If we repeat an experiment  $n$  times and observe the values  $x_1, \dots, x_n$ , the sum (3.2.9) can be interpreted as a measure of the error in assuming that the correct value is  $x$ . This exercise shows that this error is minimized by choosing  $x$  to be a median of the values  $x_1, \dots, x_n$ . The requested inductive argument is attributed to J. Lancaster.

**154.** Prove that

$$\sum_{i=1}^n \sum_{j=1}^n (i-j)^2 = \frac{n^2(n^2-1)}{6}.$$

**155.** Let  $X = \{a, b\}$ . Define a function from  $X^*$  to  $X^*$  as  $f(\alpha) = \alpha ab$ . Is  $f$  one-to-one? Is  $f$  onto  $X^*$ ? Prove your answers.

**156.** Let  $X = \{a, b\}$ . Define a function from  $X^*$  to  $X^*$  as  $f(\alpha) = \alpha\alpha$ . Is  $f$  one-to-one? Is  $f$  onto  $X^*$ ? Prove your answers.

**157.** Let  $X = \{a, b\}$ . A *palindrome over  $X$*  is a string  $\alpha$  for which  $\alpha = \alpha^R$  (i.e., a string that reads the same forward and backward). An example of a palindrome over  $X$  is  $bbaabb$ . Define a function from  $X^*$  to the set of palindromes over  $X$  as  $f(\alpha) = \alpha\alpha^R$ . Is  $f$  one-to-one? Is  $f$  onto? Prove your answers.

Let  $L$  be the set of all strings, including the null string, that can be constructed by repeated application of the following rules:

- If  $\alpha \in L$ , then  $a\alpha \in L$  and  $\alpha a \in L$ .
- If  $\alpha \in L$  and  $\beta \in L$ , then  $\alpha\beta \in L$ .

For example,  $ab$  is in  $L$ , for if we take  $\alpha = \lambda$ , then  $\alpha \in L$  and the first rule states that  $ab = a\alpha \in L$ . Similarly,  $ba \in L$ . As another example,  $aabb$  is in  $L$ , for if we take  $\alpha = ab$ , then  $\alpha \in L$ ; by the first rule,  $aabb = a\alpha b \in L$ . As a final example,  $aabbba$  is in  $L$ , for if we take  $\alpha = aabb$  and  $\beta = ba$ , then  $\alpha \in L$  and  $\beta \in L$ ; by the second rule,  $aabbba = \alpha\beta \in L$ .

**158.** Show that  $aaabbb$  is in  $L$ .

**159.** Show that  $baabab$  is in  $L$ .

**160.** Show that  $aab$  is not in  $L$ .

**161.** Prove that if  $\alpha \in L$ ,  $\alpha$  has equal numbers of  $a$ 's and  $b$ 's.

**\*162.** Prove that if  $\alpha$  has equal numbers of  $a$ 's and  $b$ 's, then  $\alpha \in L$ .

**163.** Let  $\{a_n\}_{n=1}^\infty$  be a nondecreasing sequence, which is bounded above, and let  $L$  be the least upper bound of the set  $\{a_n \mid n = 1, 2, \dots\}$ . Prove that for every real number  $\varepsilon > 0$ , there exists a positive integer  $N$  such that  $L - \varepsilon < a_n \leq L$  for every  $n \geq N$ . In calculus terminology, a nondecreasing sequence, which is bounded above, converges to the limit  $L$ , where  $L$  is the least upper bound of the set of elements of the sequence.

## 3.3 Relations

### Go Online

For more on relations, see  
[goo.gl/V3y4pS](http://goo.gl/V3y4pS)

A **relation** from one set to another can be thought of as a table that lists which elements of the first set relate to which elements of the second set (see Table 3.3.1). Table 3.3.1 shows which students are taking which courses. For example, Bill is taking Computer Science and Art, and Mary is taking Mathematics. In the terminology of relations, we would say that Bill is related to Computer Science and Art, and that Mary is related to Mathematics.

Of course, Table 3.3.1 is really just a set of ordered pairs. Abstractly, we *define* a relation to be a set of ordered pairs. In this setting, we consider the first element of the ordered pair to be related to the second element of the ordered pair.

**TABLE 3.3.1 ■ Relation of Students to Courses**

| Student | Course  |
|---------|---------|
| Bill    | CompSci |
| Mary    | Math    |
| Bill    | Art     |
| Beth    | History |
| Beth    | CompSci |
| Dave    | Math    |

**Definition 3.3.1 ►** A (*binary*) *relation R* from a set *X* to a set *Y* is a subset of the Cartesian product  $X \times Y$ . If  $(x, y) \in R$ , we write  $x R y$  and say that *x* is *related to y*. If  $X = Y$ , we call *R* a (*binary*) *relation on X*.

A function (see Section 3.1) is a special type of relation. A function *f* from *X* to *Y* is a relation from *X* to *Y* having the properties:

- (a) The domain of *f* is equal to *X*.
- (b) For each  $x \in X$ , there is exactly one  $y \in Y$  such that  $(x, y) \in f$ .

### Example 3.3.2

If we let  $X = \{\text{Bill, Mary, Beth, Dave}\}$  and  $Y = \{\text{CompSci, Math, Art, History}\}$ , our relation *R* of Table 3.3.1 can be written

$$R = \{(\text{Bill, CompSci}), (\text{Mary, Math}), (\text{Bill, Art}), (\text{Beth, History}), \\ (\text{Beth, CompSci}), (\text{Dave, Math})\}.$$

Since  $(\text{Beth, History}) \in R$ , we may write  $\text{Beth} R \text{History}$ .

Example 3.3.2 shows that a relation can be given by simply specifying which ordered pairs belong to the relation. Our next example shows that sometimes it is possible to define a relation by giving a rule for membership in the relation.

### Example 3.3.3

Let  $X = \{2, 3, 4\}$  and  $Y = \{3, 4, 5, 6, 7\}$ . If we define a relation *R* from *X* to *Y* by

$$(x, y) \in R \quad \text{if } x \text{ divides } y,$$

we obtain

$$R = \{(2, 4), (2, 6), (3, 3), (3, 6), (4, 4)\}.$$

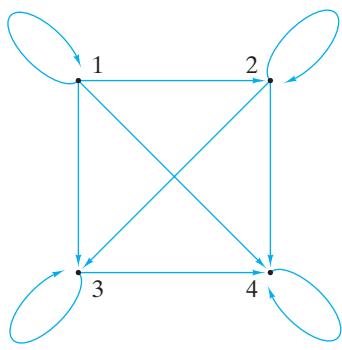
If we rewrite *R* as a table, we obtain

| X | Y |
|---|---|
| 2 | 4 |
| 2 | 6 |
| 3 | 3 |
| 3 | 6 |
| 4 | 4 |

**Example 3.3.4**

Let  $R$  be the relation on  $X = \{1, 2, 3, 4\}$  defined by  $(x, y) \in R$  if  $x \leq y$ ,  $x, y \in X$ . Then

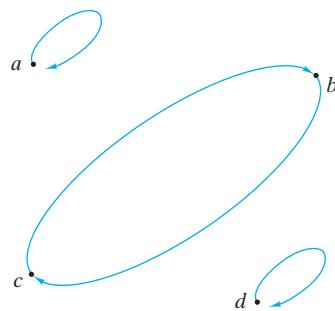
$$R = \{(1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (2, 3), (2, 4), (3, 3), (3, 4), (4, 4)\}.$$



**Figure 3.3.1** The digraph of the relation of Example 3.3.4.

**Example 3.3.5**

The relation  $R$  on  $X = \{a, b, c, d\}$  given by the digraph of Figure 3.3.2 is  $R = \{(a, a), (b, c), (c, b), (d, d)\}$ .



**Figure 3.3.2** The digraph of the relation of Example 3.3.5.

We next define several properties that relations may have.

**Definition 3.3.6** ► A relation  $R$  on a set  $X$  is *reflexive* if  $(x, x) \in R$  for every  $x \in X$ .

**Example 3.3.7**

The relation  $R$  on  $X = \{1, 2, 3, 4\}$  defined by  $(x, y) \in R$  if  $x \leq y$ ,  $x, y \in X$ , is reflexive because for each element  $x \in X$ ,  $(x, x) \in R$ ; specifically,  $(1, 1)$ ,  $(2, 2)$ ,  $(3, 3)$ , and  $(4, 4)$  are each in  $R$ . The digraph of a reflexive relation has a loop at every vertex. Notice that the digraph of this relation (see Figure 3.3.1) has a loop at every vertex.

By the generalized De Morgan's laws for logic (Theorem 1.5.14), a relation  $R$  on  $X$  is *not* reflexive if there exists  $x \in X$  such that  $(x, x) \notin R$ .

**Example 3.3.8**

The relation  $R = \{(a, a), (b, c), (c, b), (d, d)\}$  on  $X = \{a, b, c, d\}$  is not reflexive. For example,  $b \in X$ , but  $(b, b) \notin R$ . That this relation is not reflexive can also be seen by looking at its digraph (see Figure 3.3.2); vertex  $b$  does not have a loop. 

**Definition 3.3.9** ► A relation  $R$  on a set  $X$  is *symmetric* if for all  $x, y \in X$ , if  $(x, y) \in R$ , then  $(y, x) \in R$ . 

**Example 3.3.10**

The relation  $R = \{(a, a), (b, c), (c, b), (d, d)\}$  on  $X = \{a, b, c, d\}$  is symmetric because for all  $x, y$ , if  $(x, y) \in R$ , then  $(y, x) \in R$ . For example,  $(b, c)$  is in  $R$  and  $(c, b)$  is also in  $R$ . The digraph of a symmetric relation has the property that whenever there is a directed edge from  $v$  to  $w$ , there is also a directed edge from  $w$  to  $v$ . Notice that the digraph of this relation (see Figure 3.3.2) has the property that for every directed edge from  $v$  to  $w$ , there is also a directed edge from  $w$  to  $v$ . 

In symbols, a relation  $R$  is symmetric if

$$\forall x \forall y [(x, y) \in R] \rightarrow [(y, x) \in R].$$

Thus  $R$  is *not* symmetric if

$$\neg[\forall x \forall y [(x, y) \in R] \rightarrow [(y, x) \in R]]. \quad (3.3.1)$$

Using the generalized De Morgan's laws for logic (Theorem 1.5.14) and the fact that  $\neg(p \rightarrow q) \equiv p \wedge \neg q$  (see Example 1.3.13), we find that (3.3.1) is equivalent to

$$\exists x \exists y [(x, y) \in R] \wedge \neg[(y, x) \in R]$$

or, equivalently,

$$\exists x \exists y [(x, y) \in R] \wedge [(y, x) \notin R].$$

In words, a relation  $R$  is not symmetric if there exist  $x$  and  $y$  such that  $(x, y)$  is in  $R$  and  $(y, x)$  is not in  $R$ .

**Example 3.3.11**

The relation  $R$  on  $X = \{1, 2, 3, 4\}$  defined by  $(x, y) \in R$  if  $x \leq y$ ,  $x, y \in X$ , is not symmetric. For example,  $(2, 3) \in R$ , but  $(3, 2) \notin R$ . The digraph of this relation (see Figure 3.3.1) has a directed edge from 2 to 3, but there is no directed edge from 3 to 2. 

**Definition 3.3.12** ► A relation  $R$  on a set  $X$  is *antisymmetric* if for all  $x, y \in X$ , if  $(x, y) \in R$  and  $(y, x) \in R$ , then  $x = y$ . 

**Example 3.3.13**

The relation  $R$  on  $X = \{1, 2, 3, 4\}$  defined by  $(x, y) \in R$  if  $x \leq y$ ,  $x, y \in X$ , is antisymmetric because for all  $x, y$ , if  $(x, y) \in R$  (i.e.,  $x \leq y$ ) and  $(y, x) \in R$  (i.e.,  $y \leq x$ ), then  $x = y$ . 

**Example 3.3.14**

It is sometimes more convenient to replace

$$\text{if } (x, y) \in R \text{ and } (y, x) \in R, \text{ then } x = y$$

in the definition of "antisymmetric" (Definition 3.3.12) with its logically equivalent contrapositive (see Theorem 1.3.18)

if  $x \neq y$ , then  $(x, y) \notin R$  or  $(y, x) \notin R$

to obtain a logically equivalent characterization of “antisymmetric”: A relation  $R$  on a set  $X$  is antisymmetric if for all  $x, y \in X$ , if  $x \neq y$ , then  $(x, y) \notin R$  or  $(y, x) \notin R$ .

Using this equivalent definition of “antisymmetric,” we again see that the relation  $R$  on  $X = \{1, 2, 3, 4\}$  defined by  $(x, y) \in R$  if  $x \leq y$ ,  $x, y \in X$ , is antisymmetric because for all  $x, y$ , if  $x \neq y$ ,  $(x, y) \notin R$  (i.e.,  $x > y$ ) or  $(y, x) \notin R$  (i.e.,  $y > x$ ).

The equivalent characterization of “antisymmetric” translates for digraphs as follows. The digraph of an antisymmetric relation has the property that between any two distinct vertices there is at most one directed edge. Notice that the digraph of the relation  $R$  in the previous paragraph (see Figure 3.3.1) has at most one directed edge between each pair of vertices. ◀

### Example 3.3.15

If a relation has *no* members of the form  $(x, y)$ ,  $x \neq y$ , we see that the equivalent characterization of “antisymmetric”

for all  $x, y \in X$ , if  $x \neq y$ , then  $(x, y) \notin R$  or  $(y, x) \notin R$

(see Example 3.3.14) is trivially true (since the hypothesis  $x \neq y$  is always false). Thus if a relation  $R$  has *no* members of the form  $(x, y)$ ,  $x \neq y$ ,  $R$  is antisymmetric. For example,  $R = \{(a, a), (b, b), (c, c)\}$  on  $X = \{a, b, c\}$  is antisymmetric. The digraph of  $R$  shown in Figure 3.3.3 has at most one directed edge between each pair of distinct vertices. Notice that  $R$  is also reflexive and symmetric. This example shows that “antisymmetric” is not the same as “not symmetric” because this relation is in fact both symmetric and antisymmetric. ◀



**Figure 3.3.3** The digraph of the relation of Example 3.3.15.

In symbols, a relation  $R$  is antisymmetric if

$$\forall x \forall y [(x, y) \in R \wedge (y, x) \in R] \rightarrow [x = y].$$

Thus  $R$  is *not* antisymmetric if

$$\neg [\forall x \forall y [(x, y) \in R \wedge (y, x) \in R] \rightarrow [x = y]]. \quad (3.3.2)$$

Using the generalized De Morgan’s laws for logic (Theorem 1.5.14) and the fact that  $\neg(p \rightarrow q) \equiv p \wedge \neg q$  (see Example 1.3.13), we find that (3.3.2) is equivalent to

$$\exists x \exists y [(x, y) \in R \wedge (y, x) \in R] \wedge \neg[x = y]$$

which, in turn, is equivalent to

$$\exists x \exists y [(x, y) \in R \wedge (y, x) \in R \wedge (x \neq y)].$$

In words, a relation  $R$  is not antisymmetric if there exist  $x$  and  $y$ ,  $x \neq y$ , such that  $(x, y)$  and  $(y, x)$  are both in  $R$ .

### Example 3.3.16

The relation  $R = \{(a, a), (b, c), (c, b), (d, d)\}$  on  $X = \{a, b, c, d\}$  is not antisymmetric because both  $(b, c)$  and  $(c, b)$  are in  $R$ . Notice that in the digraph of this relation (see Figure 3.3.2) there are two directed edges between  $b$  and  $c$ . ◀

**Definition 3.3.17 ▶** A relation  $R$  on a set  $X$  is *transitive* if for all  $x, y, z \in X$ , if  $(x, y)$  and  $(y, z) \in R$ , then  $(x, z) \in R$ . ◀

### Example 3.3.18

The relation  $R$  on  $X = \{1, 2, 3, 4\}$  defined by  $(x, y) \in R$  if  $x \leq y$ ,  $x, y \in X$ , is transitive because for all  $x, y, z$ , if  $(x, y)$  and  $(y, z) \in R$ , then  $(x, z) \in R$ . To formally verify that this

relation satisfies Definition 3.3.17, we can list all pairs of the form  $(x, y)$  and  $(y, z)$  in  $R$  and then verify that in every case,  $(x, z) \in R$ :

| Pairs of Form |          | Pairs of Form |          |
|---------------|----------|---------------|----------|
| $(x, y)$      | $(y, z)$ | $(x, z)$      | $(x, y)$ |
| (1, 1)        | (1, 1)   | (1, 1)        | (2, 2)   |
| (1, 1)        | (1, 2)   | (1, 2)        | (2, 2)   |
| (1, 1)        | (1, 3)   | (1, 3)        | (2, 2)   |
| (1, 1)        | (1, 4)   | (1, 4)        | (2, 3)   |
| (1, 2)        | (2, 2)   | (1, 2)        | (2, 4)   |
| (1, 2)        | (2, 3)   | (1, 3)        | (2, 3)   |
| (1, 2)        | (2, 4)   | (1, 4)        | (3, 3)   |
| (1, 3)        | (3, 3)   | (1, 3)        | (3, 4)   |
| (1, 3)        | (3, 4)   | (1, 4)        | (4, 4)   |
| (1, 4)        | (4, 4)   | (1, 4)        | (4, 4)   |

Actually, some of the entries in the preceding table were unnecessary. If  $x = y$  or  $y = z$ , we need not explicitly verify that the condition

if  $(x, y)$  and  $(y, z) \in R$ , then  $(x, z) \in R$

is satisfied since it will automatically be true. Suppose, for example, that  $x = y$  and  $(x, y)$  and  $(y, z)$  are in  $R$ . Since  $x = y$ ,  $(x, z) = (y, z)$  is in  $R$  and the condition is satisfied. Eliminating the cases  $x = y$  and  $y = z$  leaves only the following to be explicitly checked to verify that the relation is transitive:

| Pairs of Form |          |          |
|---------------|----------|----------|
| $(x, y)$      | $(y, z)$ | $(x, z)$ |
| (1, 2)        | (2, 3)   | (1, 3)   |
| (1, 2)        | (2, 4)   | (1, 4)   |
| (1, 3)        | (3, 4)   | (1, 4)   |
| (2, 3)        | (3, 4)   | (2, 4)   |

The digraph of a transitive relation has the property that whenever there are directed edges from  $x$  to  $y$  and from  $y$  to  $z$ , there is also a directed edge from  $x$  to  $z$ . Notice that the digraph of this relation (see Figure 3.3.1) has this property. 

In symbols, a relation  $R$  is transitive if

$$\forall x \forall y \forall z [(x, y) \in R \wedge (y, z) \in R] \rightarrow [(x, z) \in R].$$

Thus  $R$  is *not* transitive if

$$\neg \lceil \forall x \forall y \forall z [(x, y) \in R \wedge (y, z) \in R] \rightarrow [(x, z) \in R] \rceil. \quad (3.3.3)$$

Using the generalized De Morgan's laws for logic (Theorem 1.5.14) and the fact that  $\neg(p \rightarrow q) \equiv p \wedge \neg q$  (see Example 1.3.13), we find that (3.3.3) is equivalent to

$$\exists x \exists y \exists z [(x, y) \in R \wedge (y, z) \in R] \wedge \neg [(x, z) \in R]$$

or, equivalently,

$$\exists x \exists y \exists z [(x, y) \in R \wedge (y, z) \in R \wedge (x, z) \notin R].$$

In words, a relation  $R$  is not transitive if there exist  $x$ ,  $y$ , and  $z$  such that  $(x, y)$  and  $(y, z)$  are in  $R$ , but  $(x, z)$  is not in  $R$ .

**Example 3.3.19**

The relation  $R = \{(a, a), (b, c), (c, b), (d, d)\}$  on  $X = \{a, b, c, d\}$  is not transitive. For example,  $(b, c)$  and  $(c, b)$  are in  $R$ , but  $(b, b)$  is not in  $R$ . Notice that in the digraph of this relation (see Figure 3.3.2) there are directed edges from  $b$  to  $c$  and from  $c$  to  $b$ , but there is no directed edge from  $b$  to  $b$ . 

Relations can be used to order elements of a set. For example, the relation  $R$  defined on the set of integers by

$$(x, y) \in R \quad \text{if } x \leq y$$

orders the integers. Notice that the relation  $R$  is reflexive, antisymmetric, and transitive. Such relations are called **partial orders**.

**Definition 3.3.20** ► A relation  $R$  on a set  $X$  is a *partial order* if  $R$  is reflexive, antisymmetric, and transitive. 

**Example 3.3.21**

Since the relation  $R$  defined on the positive integers by

$$(x, y) \in R \quad \text{if } x \text{ divides } y$$

is reflexive, antisymmetric, and transitive,  $R$  is a partial order. 

If  $R$  is a partial order on a set  $X$ , the notation  $x \preceq y$  is sometimes used to indicate that  $(x, y) \in R$ . This notation suggests that we are interpreting the relation as an ordering of the elements in  $X$ .

Suppose that  $R$  is a partial order on a set  $X$ . If  $x, y \in X$  and either  $x \preceq y$  or  $y \preceq x$ , we say that  $x$  and  $y$  are **comparable**. If  $x, y \in X$  and  $x \not\preceq y$  and  $y \not\preceq x$ , we say that  $x$  and  $y$  are **incomparable**. If every pair of elements in  $X$  is comparable, we call  $R$  a **total order**. The less than or equal to relation on the positive integers is a total order since, if  $x$  and  $y$  are integers, either  $x \leq y$  or  $y \leq x$ . The reason for the term “partial order” is that in general some elements in  $X$  may be incomparable. The “divides” relation on the positive integers (see Example 3.3.21) has both comparable and incomparable elements. For example, 2 and 3 are incomparable (since 2 does not divide 3 and 3 does not divide 2), but 3 and 6 are comparable (since 3 divides 6).

One application of partial orders is to task scheduling.

**Example 3.3.22**

**Task Scheduling** Consider the set  $T$  of tasks that must be completed in order to take an indoor flash picture with a particular camera.

1. Remove lens cap.
2. Focus camera.
3. Turn off safety lock.
4. Turn on flash unit.
5. Push photo button.

Some of these tasks must be done before others. For example, task 1 must be done before task 2. On the other hand, other tasks can be done in either order. For example, tasks 2 and 3 can be done in either order.

The relation  $R$  defined on  $T$  by

$$iRj \quad \text{if } i = j \text{ or task } i \text{ must be done before task } j$$

orders the tasks. We obtain

$$R = \{(1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (1, 2), (1, 5), (2, 5), (3, 5), (4, 5)\}.$$

Since  $R$  is reflexive, antisymmetric, and transitive, it is a partial order. A solution to the problem of scheduling the tasks so that we can take a picture is a total ordering of the tasks consistent with the partial order. More precisely, we require a total ordering of the tasks  $t_1, t_2, t_3, t_4, t_5$  such that if  $t_i R t_j$ , then  $i = j$  or  $t_i$  precedes  $t_j$  in the list. Among the solutions are  $1, 2, 3, 4, 5$  and  $3, 4, 1, 2, 5$ . 

Given a relation  $R$  from  $X$  to  $Y$ , we may define a relation from  $Y$  to  $X$  by reversing the order of each ordered pair in  $R$ . The inverse relation generalizes the inverse function. The formal definition follows.

**Definition 3.3.23** ► Let  $R$  be a relation from  $X$  to  $Y$ . The *inverse* of  $R$ , denoted  $R^{-1}$ , is the relation from  $Y$  to  $X$  defined by

$$R^{-1} = \{(y, x) \mid (x, y) \in R\}. \quad \blacktriangleleft$$

**Example 3.3.24** If we define a relation  $R$  from  $X = \{2, 3, 4\}$  to  $Y = \{3, 4, 5, 6, 7\}$  by

$$(x, y) \in R \quad \text{if } x \text{ divides } y,$$

we obtain

$$R = \{(2, 4), (2, 6), (3, 3), (3, 6), (4, 4)\}.$$

The inverse of this relation is

$$R^{-1} = \{(4, 2), (6, 2), (3, 3), (6, 3), (4, 4)\}.$$

In words, we might describe this relation as “is divisible by.” 

If we have a relation  $R_1$  from  $X$  to  $Y$  and a relation  $R_2$  from  $Y$  to  $Z$ , we can form the composition of the relations by applying first relation  $R_1$  and then relation  $R_2$ . Composition of relations generalizes composition of functions. The formal definition follows.

**Definition 3.3.25** ► Let  $R_1$  be a relation from  $X$  to  $Y$  and  $R_2$  be a relation from  $Y$  to  $Z$ . The *composition* of  $R_1$  and  $R_2$ , denoted  $R_2 \circ R_1$ , is the relation from  $X$  to  $Z$  defined by

$$R_2 \circ R_1 = \{(x, z) \mid (x, y) \in R_1 \text{ and } (y, z) \in R_2 \text{ for some } y \in Y\}. \quad \blacktriangleleft$$

**Example 3.3.26** The composition of the relations

$$R_1 = \{(1, 2), (1, 6), (2, 4), (3, 4), (3, 6), (3, 8)\}$$

and

$$R_2 = \{(2, u), (4, s), (4, t), (6, t), (8, u)\}$$

is

$$R_2 \circ R_1 = \{(1, u), (1, t), (2, s), (2, t), (3, s), (3, t), (3, u)\}.$$

For example,  $(1, u) \in R_2 \circ R_1$  because  $(1, 2) \in R_1$  and  $(2, u) \in R_2$ . 

**Example 3.3.27** Suppose that  $R$  and  $S$  are transitive relations on a set  $X$ . Determine whether each of  $R \cup S$ ,  $R \cap S$ , or  $R \circ S$  must be transitive.

**SOLUTION** We try to prove each of the three statements. If we fail, we will try to determine where our proof fails and use this information to construct a counterexample.

To prove that  $R \cup S$  is transitive, we must show that if  $(x, y), (y, z) \in R \cup S$ , then  $(x, z) \in R \cup S$ . Suppose that  $(x, y), (y, z) \in R \cup S$ . If  $(x, y)$  and  $(y, z)$  happen to both be in  $R$ , we could use the fact that  $R$  is transitive to conclude that  $(x, z) \in R$  and, therefore,  $(x, z) \in R \cup S$ . A similar argument shows that if  $(x, y)$  and  $(y, z)$  happen to both be in  $S$ , then  $(x, z) \in R \cup S$ . But what if  $(x, y) \in R$  and  $(y, z) \in S$ ? Now the fact that  $R$  and  $S$  are transitive seems to be of no help. We try to construct a counterexample in which  $R$  and  $S$  are transitive, but there exist  $(x, y) \in R$  and  $(y, z) \in S$  such that  $(x, z) \notin R \cup S$ .

We put  $(1, 2)$  in  $R$  and  $(2, 3)$  in  $S$  and ensure that  $(1, 3)$  is not in  $R \cup S$ . In fact, if  $R = \{(1, 2)\}$ ,  $R$  is transitive. Similarly, if  $S = \{(2, 3)\}$ ,  $S$  is transitive. We have our counterexample. We could write up our solution as follows.

We show that  $R \cup S$  need not be transitive. Let  $R = \{(1, 2)\}$  and  $S = \{(2, 3)\}$ . Then  $R$  and  $S$  are transitive, but  $R \cup S$  is not transitive;  $(1, 2), (2, 3) \in R \cup S$ , but  $(1, 3) \notin R \cup S$ .

Next we turn our attention to  $R \cap S$ . To prove that  $R \cap S$  is transitive, we must show that if  $(x, y), (y, z) \in R \cap S$ , then  $(x, z) \in R \cap S$ . Suppose that  $(x, y), (y, z) \in R \cap S$ . Then  $(x, y), (y, z) \in R$ . Since  $R$  is transitive,  $(x, z) \in R$ . Similarly,  $(x, y), (y, z) \in S$ , and since  $S$  is transitive,  $(x, z) \in S$ . Therefore  $(x, z) \in R \cap S$ . We have proved that  $R \cap S$  is transitive.

Finally, consider  $R \circ S$ . To prove that  $R \circ S$  is transitive, we must show that if  $(x, y), (y, z) \in R \circ S$ , then  $(x, z) \in R \circ S$ . Suppose that  $(x, y), (y, z) \in R \circ S$ . Then there exists  $a$  such that  $(x, a) \in S$  and  $(a, y) \in R$ , and there exists  $b$  such that  $(y, b) \in S$  and  $(b, z) \in R$ . We now know that  $(a, y), (b, z) \in R$ , but the fact that  $R$  is transitive does not allow us to infer anything from  $(a, y), (b, z) \in R$ . A similar statement applies to  $S$ . We try to construct a counterexample in which  $R$  and  $S$  are transitive but  $R \circ S$  is not transitive.

We will arrange for  $(1, 2), (2, 3) \in R \circ S$ , but  $(1, 3) \notin R \circ S$ . In order for  $(1, 2) \in R \circ S$ , we must have  $(1, a) \in S$  and  $(a, 2) \in R$ , for some  $a$ . We put  $(1, 5)$  in  $S$  and  $(5, 2)$  in  $R$ . (We chose  $a$  to be a number different from 1, 2, or 3 to avoid a clash with those numbers. Any number different from 1, 2, 3 would do.) So far, so good! In order for  $(2, 3) \in R \circ S$ , we must have  $(2, b) \in S$  and  $(b, 3) \in R$ , for some  $b$ . We put  $(2, 6)$  in  $S$  and  $(6, 3)$  in  $R$ . (Again, we chose  $b = 6$  to avoid a clash with the other numbers already chosen.) Now  $R = \{(5, 2), (6, 3)\}$  and  $S = \{(1, 5), (2, 6)\}$ . Notice that  $R$  and  $S$  are transitive. We have our counterexample. We could write up our solution as follows.

We show that  $R \circ S$  need not be transitive. Let  $R = \{(5, 2), (6, 3)\}$  and  $S = \{(1, 5), (2, 6)\}$ . Then  $R$  and  $S$  are transitive. Now  $R \circ S = \{(1, 2), (2, 3)\}$  is not transitive;  $(1, 2), (2, 3) \in R \circ S$ , but  $(1, 3) \notin R \circ S$ . 

### 3.3 Problem-Solving Tips

- To prove that a relation is reflexive, show that  $(x, x) \in R$  for every  $x \in X$ . In words, a relation is reflexive if every element in  $X$  is related to itself. Given an arrow diagram, the relation is reflexive if there is a loop at every vertex.
- To prove that a relation  $R$  on a set  $X$  is *not* reflexive, find  $x \in X$  such that  $(x, x) \notin R$ . Given an arrow diagram, the relation is not reflexive if some vertex has no loop.
- To prove that a relation  $R$  on a set  $X$  is symmetric, show that for all  $x, y \in X$ , if  $(x, y) \in R$ , then  $(y, x) \in R$ . In words, a relation is symmetric if whenever  $x$  is related to  $y$ , then  $y$  is related to  $x$ . Given an arrow diagram, the relation is symmetric

if whenever there is a directed edge from  $x$  to  $y$ , there is also a directed edge from  $y$  to  $x$ .

- To prove that a relation  $R$  on a set  $X$  is *not* symmetric, find  $x, y \in X$  such that  $(x, y) \in R$  and  $(y, x) \notin R$ . Given an arrow diagram, the relation is not symmetric if there are two distinct vertices  $x$  and  $y$  with a directed edge from  $x$  to  $y$  but no directed edge from  $y$  to  $x$ .
- To prove that a relation  $R$  on a set  $X$  is antisymmetric, show that for all  $x, y \in X$ , if  $(x, y) \in R$  and  $(y, x) \in R$ , then  $x = y$ . In words, a relation is antisymmetric if whenever  $x$  is related to  $y$  and  $y$  is related to  $x$ , then  $x = y$ . An equivalent characterization of “antisymmetric” can also be used: Show that for all  $x, y \in X$ , if  $x \neq y$ , then  $(x, y) \notin R$  or  $(y, x) \notin R$ . Given an arrow diagram, the relation is antisymmetric if between any two distinct vertices there is at most one directed edge. Note that “*not* symmetric” is not necessarily the same as “antisymmetric.”
- To prove that a relation  $R$  on a set  $X$  is *not* antisymmetric, find  $x, y \in X$ ,  $x \neq y$ , such that  $(x, y) \in R$  and  $(y, x) \in R$ . Given an arrow diagram, the relation is not anti-symmetric if there are two distinct vertices  $x$  and  $y$  and two directed edges, one from  $x$  to  $y$  and the other from  $y$  to  $x$ .
- To prove that a relation  $R$  on a set  $X$  is transitive, show that for all  $x, y, z \in X$ , if  $(x, y)$  and  $(y, z)$  are in  $R$ , then  $(x, z)$  is in  $R$ . [It suffices to check ordered pairs  $(x, y)$  and  $(y, z)$  with  $x \neq y$  and  $y \neq z$ .] In words, a relation is transitive if whenever  $x$  is related to  $y$  and  $y$  is related to  $z$ , then  $x$  is related to  $z$ . Given an arrow diagram, the relation is transitive if whenever there are directed edges from  $x$  to  $y$  and from  $y$  to  $z$ , there is also a directed edge from  $x$  to  $z$ .
- To prove that a relation  $R$  on a set  $X$  is *not* transitive, find  $x, y, z \in X$  such that  $(x, y)$  and  $(y, z)$  are in  $R$ , but  $(x, z)$  is not in  $R$ . Given an arrow diagram, the relation is not transitive if there are three distinct vertices  $x, y, z$  and directed edges from  $x$  to  $y$  and from  $y$  to  $z$ , but no directed edge from  $x$  to  $z$ .
- A *partial order* is a relation that is reflexive, antisymmetric, and transitive.
- The *inverse*  $R^{-1}$  of the relation  $R$  consists of the elements  $(y, x)$ , where  $(x, y) \in R$ . In words,  $x$  is related to  $y$  in  $R$  if and only if  $y$  is related to  $x$  in  $R^{-1}$ .
- If  $R_1$  is a relation from  $X$  to  $Y$  and  $R_2$  is a relation from  $Y$  to  $Z$ , the *composition* of  $R_1$  and  $R_2$ , denoted  $R_2 \circ R_1$ , is the relation from  $X$  to  $Z$  defined by

$$R_2 \circ R_1 = \{(x, z) \mid (x, y) \in R_1 \text{ and } (y, z) \in R_2 \text{ for some } y \in Y\}.$$

To compute the composition, find all pairs of the form  $(x, y) \in R_1$  and  $(y, z) \in R_2$ ; then put  $(x, z)$  in  $R_2 \circ R_1$ .

### 3.3 Review Exercises

1. What is a binary relation from  $X$  to  $Y$ ?
2. What is the digraph of a binary relation?
3. Define *reflexive relation*. Give an example of a reflexive relation. Give an example of a relation that is *not* reflexive.
4. Define *symmetric relation*. Give an example of a symmetric relation. Give an example of a relation that is *not* symmetric.
5. Define *antisymmetric relation*. Give an example of an antisymmetric relation. Give an example of a relation that is *not* anti-symmetric.
6. Define *transitive relation*. Give an example of a transitive relation. Give an example of a relation that is *not* transitive.
7. Define *partial order* and give an example of a partial order.
8. Define *inverse relation* and give an example of an inverse relation.
9. Define *composition of relations* and give an example of the composition of relations.

### 3.3 Exercises

In Exercises 1–4, write the relation as a set of ordered pairs.

|           |        |
|-----------|--------|
| <u>1.</u> |        |
| 8840      | Hammer |
| 9921      | Pliers |
| 452       | Paint  |
| 2207      | Carpet |

|           |         |
|-----------|---------|
| <u>2.</u> |         |
| Sally     | Math    |
| Ruth      | Physics |
| Sam       | Econ    |

|           |   |
|-----------|---|
| <u>3.</u> |   |
| a         | 3 |
| b         | 1 |
| b         | 4 |
| c         | 1 |

|           |   |
|-----------|---|
| <u>4.</u> |   |
| a         | a |
| b         | b |

In Exercises 5–8, write the relation as a table.

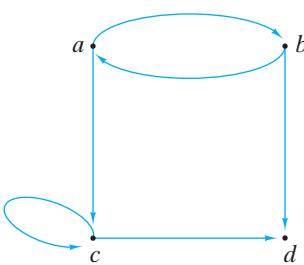
5.  $R = \{(a, 6), (b, 2), (a, 1), (c, 1)\}$
6. The relation  $R$  on  $\{1, 2, 3, 4\}$  defined by  $(x, y) \in R$  if  $x^2 \geq y$
7.  $R = \{(\text{Roger}, \text{Music}), (\text{Pat}, \text{History}), (\text{Ben}, \text{Math}), (\text{Pat}, \text{PolySci})\}$
8. The relation  $R$  from the set  $X$  of planets to the set  $Y$  of integers defined by  $(x, y) \in R$  if  $x$  is in position  $y$  from the sun (nearest the sun being in position 1, second nearest the sun being in position 2, and so on)

In Exercises 9–12, draw the digraph of the relation.

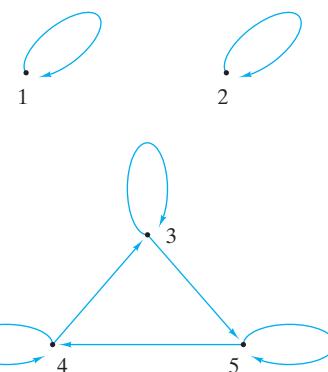
9. The relation of Exercise 4 on  $\{a, b, c\}$
10. The relation  $R = \{(1, 2), (2, 3), (3, 4), (4, 1)\}$  on  $\{1, 2, 3, 4\}$
11. The relation  $R = \{(1, 2), (2, 1), (3, 3), (1, 1), (2, 2)\}$  on  $X = \{1, 2, 3\}$
12. The relation of Exercise 6

In Exercises 13–16, write the relation as a set of ordered pairs.

13.



14.



15.

1 •      2 •

16.



17. Find the inverse (as a set of ordered pairs) of each relation in Exercises 1–16.

Exercises 18 and 19 refer to the relation  $R$  on the set  $\{1, 2, 3, 4, 5\}$  defined by the rule  $(x, y) \in R$  if 3 divides  $x - y$ .

18. List the elements of  $R$ .
19. List the elements of  $R^{-1}$ .
20. Repeat Exercises 18 and 19 for the relation  $R$  on the set  $\{1, 2, 3, 4, 5\}$  defined by the rule  $(x, y) \in R$  if  $x + y \leq 6$ .
21. Repeat Exercises 18 and 19 for the relation  $R$  on the set  $\{1, 2, 3, 4, 5\}$  defined by the rule  $(x, y) \in R$  if  $x = y - 1$ .
22. Is the relation of Exercise 20 reflexive, symmetric, antisymmetric, transitive, and/or a partial order?
23. Is the relation of Exercise 21 reflexive, symmetric, antisymmetric, transitive, and/or a partial order?

In Exercises 24–34, determine whether each relation defined on the set of positive integers is reflexive, symmetric, antisymmetric, transitive, and/or a partial order.

24.  $(x, y) \in R$  if  $xy = 1$ .
25.  $(x, y) \in R$  if  $xy = 2$ .
26.  $(x, y) \in R$  if  $xy \geq 1$ .
27.  $(x, y) \in R$  if  $x = y^2$ .
28.  $(x, y) \in R$  if  $x > y$ .
29.  $(x, y) \in R$  if  $x \geq y$ .
30.  $(x, y) \in R$  if  $x = y$ .
31.  $(x, y) \in R$  if 3 divides  $x - y$ .
32.  $(x, y) \in R$  if 3 divides  $x + 2y$ .
33.  $(x, y) \in R$  if  $x - y = 2$ .
34.  $(x, y) \in R$  if  $|x - y| = 2$ .
35. Let  $X$  be a nonempty set. Define a relation on  $\mathcal{P}(X)$ , the power set of  $X$ , as  $(A, B) \in R$  if  $A \subseteq B$ . Is this relation reflexive, symmetric, antisymmetric, transitive, and/or a partial order?

- 36.** Prove that a relation  $R$  on a set  $X$  is antisymmetric if and only if for all  $x, y \in X$ , if  $(x, y) \in R$  and  $x \neq y$ , then  $(y, x) \notin R$ .
- 37.** Let  $X$  be the set of all four-bit strings (e.g., 0011, 0101, 1000). Define a relation  $R$  on  $X$  as  $s_1 R s_2$  if some substring of  $s_1$  of length 2 is equal to some substring of  $s_2$  of length 2. Examples: 0111  $R$  1010 (because both 0111 and 1010 contain 01), 1110  $\not R$  0001 (because 1110 and 0001 do not share a common substring of length 2). Is this relation reflexive, symmetric, antisymmetric, transitive, and/or a partial order?
- 38.** Suppose that  $R_i$  is a partial order on  $X_i$ ,  $i = 1, 2$ . Show that  $R$  is a partial order on  $X_1 \times X_2$  if we define
- $$(x_1, x_2) R (x'_1, x'_2) \quad \text{if } x_1 R_1 x'_1 \text{ and } x_2 R_2 x'_2.$$
- 39.** Let  $R_1$  and  $R_2$  be the relations on  $\{1, 2, 3, 4\}$  given by
- $$R_1 = \{(1, 1), (1, 2), (3, 4), (4, 2)\}$$
- $$R_2 = \{(1, 1), (2, 1), (3, 1), (4, 4), (2, 2)\}.$$
- List the elements of  $R_1 \circ R_2$  and  $R_2 \circ R_1$ .
- Give examples of relations on  $\{1, 2, 3, 4\}$  having the properties specified in Exercises 40–44.
- 40.** Reflexive, symmetric, and not transitive
- 41.** Reflexive, not symmetric, and not transitive
- 42.** Reflexive, antisymmetric, and not transitive
- 43.** Not reflexive, symmetric, not antisymmetric, and transitive
- 44.** Not reflexive, not symmetric, and transitive
- Let  $R$  and  $S$  be relations on  $X$ . Determine whether each statement in Exercises 45–57 is true or false. If the statement is true, prove it; otherwise, give a counterexample.
- 45.** If  $R$  is transitive, then  $R^{-1}$  is transitive.
- 46.** If  $R$  and  $S$  are reflexive, then  $R \cup S$  is reflexive.
- 47.** If  $R$  and  $S$  are reflexive, then  $R \cap S$  is reflexive.
- 48.** If  $R$  and  $S$  are reflexive, then  $R \circ S$  is reflexive.
- 49.** If  $R$  is reflexive, then  $R^{-1}$  is reflexive.
- 50.** If  $R$  and  $S$  are symmetric, then  $R \cup S$  is symmetric.
- 51.** If  $R$  and  $S$  are symmetric, then  $R \cap S$  is symmetric.
- 52.** If  $R$  and  $S$  are symmetric, then  $R \circ S$  is symmetric.
- 53.** If  $R$  is symmetric, then  $R^{-1}$  is symmetric.
- 54.** If  $R$  and  $S$  are antisymmetric, then  $R \cup S$  is antisymmetric.
- 55.** If  $R$  and  $S$  are antisymmetric, then  $R \cap S$  is antisymmetric.
- 56.** If  $R$  and  $S$  are antisymmetric, then  $R \circ S$  is antisymmetric.
- 57.** If  $R$  is antisymmetric, then  $R^{-1}$  is antisymmetric.
- 58.** How many relations are there on an  $n$ -element set?
- In Exercises 59–61, determine whether each relation  $R$  defined on the collection of all nonempty subsets of real numbers is reflexive, symmetric, antisymmetric, transitive, and/or a partial order.
- 59.**  $(A, B) \in R$  if for every  $\varepsilon > 0$ , there exists  $a \in A$  and  $b \in B$  with  $|a - b| < \varepsilon$ .
- 60.**  $(A, B) \in R$  if for every  $a \in A$  and  $\varepsilon > 0$ , there exists  $b \in B$  with  $|a - b| < \varepsilon$ .
- 61.**  $(A, B) \in R$  if for every  $a \in A$ ,  $b \in B$ , and  $\varepsilon > 0$ , there exists  $a' \in A$  and  $b' \in B$  with  $|a - b'| < \varepsilon$  and  $|a' - b| < \varepsilon$ .
- 62.** What is wrong with the following argument, which supposedly shows that any relation  $R$  on  $X$  that is symmetric and transitive is reflexive?
- Let  $x \in X$ . Using symmetry, we have  $(x, y)$  and  $(y, x)$  both in  $R$ . Since  $(x, y), (y, x) \in R$ , by transitivity we have  $(x, x) \in R$ . Therefore,  $R$  is reflexive.

## 3.4 Equivalence Relations

### Go Online

For more on equivalence relations, see  
[goo.gl/V3y4pS](http://goo.gl/V3y4pS)

Suppose that we have a set  $X$  of 10 balls, each of which is either red, blue, or green (see Figure 3.4.1). If we divide the balls into sets  $R$ ,  $B$ , and  $G$  according to color, the family  $\{R, B, G\}$  is a partition of  $X$ . (Recall that in Section 1.1, we defined a partition of a set  $X$  to be a collection  $\mathcal{S}$  of nonempty subsets of  $X$  such that every element in  $X$  belongs to exactly one member of  $\mathcal{S}$ .)

A partition can be used to define a relation. If  $\mathcal{S}$  is a partition of  $X$ , we may define  $x R y$  to mean that for some set  $S \in \mathcal{S}$ , both  $x$  and  $y$  belong to  $S$ . For the example of Figure 3.4.1, the relation obtained could be described as “is the same color as.” The next theorem shows that such a relation is always reflexive, symmetric, and transitive.

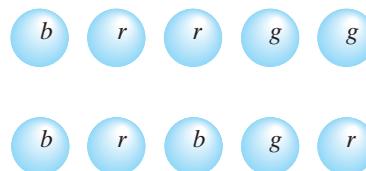


Figure 3.4.1 A set of colored balls.

If  $n$  is an arbitrary positive integer (not necessarily a power of 2), it lies between two powers of 2; that is, for some  $k \geq 1$ ,

$$2^{k-1} \leq n < 2^k.$$

We use induction on  $k$  to show that in this case the statement  $x = x + 1$  is executed  $k$  times.

If  $k = 1$ , we have

$$1 = 2^{1-1} \leq n < 2^1 = 2.$$

Therefore,  $n$  is 1. In this case, the statement  $x = x + 1$  is executed once. Thus the Basis Step is proved.

Now suppose that if  $n$  satisfies

$$2^{k-1} \leq n < 2^k, \quad (4.3.3)$$

the statement  $x = x + 1$  is executed  $k$  times. We must show that if  $n$  satisfies

$$2^k \leq n < 2^{k+1}, \quad (4.3.4)$$

the statement  $x = x + 1$  is executed  $k + 1$  times.

Suppose that  $n$  satisfies (4.3.4). At line 1,  $i$  is set to  $n$ . At line 2, the condition  $i \geq 1$  is true. At line 3, we execute the statement  $x = x + 1$  the first time. At line 4,  $i$  is reset to  $\lfloor n/2 \rfloor$  and we return to line 2. Notice that

$$2^{k-1} \leq n/2 < 2^k.$$

Because  $2^{k-1}$  is an integer, we must also have

$$2^{k-1} \leq \lfloor n/2 \rfloor < 2^k.$$

By the inductive assumption (4.3.3), the statement  $x = x + 1$  is executed  $k$  more times, for a total of  $k + 1$  times. The Inductive Step is complete. Therefore, if  $n$  satisfies (4.3.3), the statement  $x = x + 1$  is executed  $k$  times.

Suppose that  $n$  satisfies (4.3.3). Taking logarithms to the base 2, we have

$$k - 1 \leq \lg n < k.$$

Therefore,  $k$ , the number of times the statement  $x = x + 1$  is executed, satisfies

$$\lg n < k \leq 1 + \lg n.$$

Because  $k$  is an integer, we must have  $k \leq 1 + \lfloor \lg n \rfloor$ . Furthermore,  $\lfloor \lg n \rfloor < k$ . It follows from the last two inequalities that  $k = 1 + \lfloor \lg n \rfloor$ . Since  $1 + \lfloor \lg n \rfloor = \Theta(\lg n)$ , a theta notation for the number of times the statement  $x = x + 1$  is executed is  $\Theta(\lg n)$ . 

Many algorithms are based on the idea of repeated halving. Example 4.3.14 shows that for size  $n$ , repeated halving takes time  $\Theta(\lg n)$ . Of course, the algorithm may do work in addition to the halving that will increase the overall time.

#### **Example 4.3.15**

Find a theta notation in terms of  $n$  for the number of times the statement  $x = x + 1$  is executed.

```

1. $j = n$
2. while ($j \geq 1$) {
3. for $i = 1$ to j
4. $x = x + 1$
5. $j = \lfloor j/2 \rfloor$
6. }

```

**SOLUTION** Let  $t(n)$  denote the number of times we execute the statement  $x = x + 1$ . The first time we arrive at the body of the while loop, the statement  $x = x + 1$  is executed  $n$  times. Therefore  $t(n) \geq n$  for all  $n \geq 1$  and  $t(n) = \Omega(n)$ .

Next we derive a big oh notation for  $t(n)$ . After  $j$  is set to  $n$ , we arrive at the while loop for the first time. The statement  $x = x + 1$  is executed  $n$  times. At line 5,  $j$  is replaced by  $\lfloor n/2 \rfloor$ ; hence  $j \leq n/2$ . If  $j \geq 1$ , we will execute  $x = x + 1$  at most  $n/2$  additional times in the next iteration of the while loop, and so on. If we let  $k$  denote the number of times we execute the body of the while loop, the number of times we execute  $x = x + 1$  is at most

$$n + \frac{n}{2} + \frac{n}{4} + \cdots + \frac{n}{2^{k-1}}.$$

This geometric sum (see Example 2.4.4) is equal to

$$\frac{n(1 - \frac{1}{2^k})}{1 - \frac{1}{2}}.$$

Now

$$t(n) \leq \frac{n(1 - \frac{1}{2^k})}{1 - \frac{1}{2}} = 2n \left(1 - \frac{1}{2^k}\right) \leq 2n \quad \text{for all } n \geq 1,$$

so  $t(n) = O(n)$ . Thus a theta notation for the number of times we execute  $x = x + 1$  is  $\Theta(n)$ . 

### Example 4.3.16

Determine, in theta notation, the best-case, worst-case, and average-case times required to execute Algorithm 4.3.17, which follows. Assume that the input size is  $n$  and that the run time of the algorithm is the number of comparisons made at line 3. Also, assume that the  $n + 1$  possibilities of  $key$  being at any particular position in the sequence or not being in the sequence are equally likely.

**SOLUTION** The best-case time can be analyzed as follows. If  $s_1 = key$ , line 3 is executed once. Thus the best-case time of Algorithm 4.3.17 is  $\Theta(1)$ .

The worst-case time of Algorithm 4.3.17 is analyzed as follows. If  $key$  is not in the sequence, line 3 will be executed  $n$  times, so the worst-case time of Algorithm 4.3.17 is  $\Theta(n)$ .

Finally, consider the average-case time of Algorithm 4.3.17. If  $key$  is found at the  $i$ th position, line 3 is executed  $i$  times; if  $key$  is not in the sequence, line 3 is executed  $n$  times. Thus the average number of times line 3 is executed is

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

Now

$$\begin{aligned} \frac{(1 + 2 + \cdots + n) + n}{n + 1} &\leq \frac{n^2 + n}{n + 1} && \text{by (4.3.1)} \\ &= \frac{n(n + 1)}{n + 1} = n. \end{aligned}$$

Therefore, the average-case time of Algorithm 4.3.17 is  $O(n)$ . Also,

$$\begin{aligned} \frac{(1+2+\cdots+n)+n}{n+1} &\geq \frac{n^2/4+n}{n+1} && \text{by (4.3.2)} \\ &\geq \frac{n^2/4+n/4}{n+1} = \frac{n}{4}. \end{aligned}$$

Therefore the average-case time of Algorithm 4.3.17 is  $\Omega(n)$ . Thus the average-case time of Algorithm 4.3.17 is  $\Theta(n)$ . For this algorithm, the worst-case and average-case times are both  $\Theta(n)$ . ◀

### Algorithm 4.3.17

### Searching an Unordered Sequence

Given the sequence  $s_1, \dots, s_n$  and a value  $key$ , this algorithm returns the index of  $key$ . If  $key$  is not found, the algorithm returns 0.

Input:  $s_1, s_2, \dots, s_n, n$ , and  $key$  (the value to search for)  
Output: The index of  $key$ , or if  $key$  is not found, 0

1. *linear\_search(s, n, key) {*
2.     *for i = 1 to n*
3.         *if (key == s<sub>i</sub>)*
4.             *return i // successful search*
5.         *return 0 // unsuccessful search*
6. *}*

### Example 4.3.18

**Matrix Multiplication and Transitive Relations** If  $A$  is a matrix, we let  $A_{ij}$  denote the entry in row  $i$ , column  $j$ . The product of  $n \times n$  matrices  $A$  and  $B$  (i.e.,  $A$  and  $B$  have  $n$  rows and  $n$  columns) is defined as the  $n \times n$  matrix  $C$ , where

$$C_{ij} = \sum_{k=1}^n A_{ik}B_{kj} \quad 1 \leq i \leq n, 1 \leq j \leq n.$$

Algorithm 4.3.19, which computes the matrix product, is a direct translation of the preceding definition. Because of the nested loops, it runs in time  $\Theta(n^3)$ .

Recall (see the discussion following Theorem 3.5.6) that we can test whether a relation  $R$  on an  $n$ -element set is transitive by squaring its adjacency matrix, say  $A$ , and then comparing  $A^2$  with  $A$ . The relation  $R$  is transitive if and only if, whenever the entry in row  $i$ , column  $j$  in  $A^2$  is nonzero, the corresponding entry in  $A$  is also nonzero. Since there are  $n^2$  entries in  $A$  and  $A^2$ , the worst-case time to compare the entries is  $\Theta(n^2)$ . Using Algorithm 4.3.19 to compute  $A^2$  requires time  $\Theta(n^3)$ . Therefore, the overall time to test whether a relation on an  $n$ -element set is transitive, using Algorithm 4.3.19 to compute  $A^2$ , is  $\Theta(n^3)$ .

For many years it was believed that the minimum time to multiply two  $n \times n$  matrices was  $\Theta(n^3)$ ; thus it was quite a surprise when a more efficient algorithm was discovered. Strassen's algorithm (see [Johnsonbaugh: Section 5.4]) to multiply two  $n \times n$  matrices runs in time  $\Theta(n^{\lg 7})$ . Since  $\lg 7$  is approximately 2.807, Strassen's algorithm runs in time approximately  $\Theta(n^{2.807})$ , which is asymptotically faster than Algorithm 4.3.19. An algorithm by Coppersmith and Winograd (see [Coppersmith]) runs in time  $\Theta(n^{2.376})$  and, so, is even asymptotically faster than Strassen's algorithm. Since the product of two  $n \times n$  matrices contains  $n^2$  terms, *any* algorithm that multiplies two  $n \times n$  matrices requires time at least  $\Omega(n^2)$ . At the present time, no sharper lower bound is known. ◀

**Algorithm 4.3.19****Matrix Multiplication**

This algorithm computes the product  $C$  of the  $n \times n$  matrices  $A$  and  $B$  directly from the definition of matrix multiplication.

```

Input: A, B, n
Output: C , the product of A and B

matrix_product(A, B, n) {
 for $i = 1$ to n
 for $j = 1$ to n {
 $C_{ij} = 0$
 for $k = 1$ to n
 $C_{ij} = C_{ij} + A_{ik} * B_{kj}$
 }
 return C
}

```

The constants that are suppressed in the theta notation may be important. Even if for any input of size  $n$ , algorithm  $A$  requires exactly  $C_1n$  time units and algorithm  $B$  requires exactly  $C_2n^2$  time units, for certain sizes of inputs algorithm  $B$  may be superior. For example, suppose that for any input of size  $n$ , algorithm  $A$  requires  $300n$  units of time and algorithm  $B$  requires  $5n^2$  units of time. For an input size of  $n = 5$ , algorithm  $A$  requires 1500 units of time and algorithm  $B$  requires 125 units of time, and thus algorithm  $B$  is faster. Of course, for sufficiently large inputs, algorithm  $A$  is considerably faster than algorithm  $B$ .

A real-world example of the importance of constants in the theta notation is provided by matrix multiplication. Algorithm 4.3.19, which runs in time  $\Theta(n^3)$ , is typically used to multiply matrices even though the Strassen and Coppersmith-Winograd algorithms (see Example 4.3.18), which run in times  $\Theta(n^{2.807})$  and  $\Theta(n^{2.376})$ , are asymptotically faster. The constants in the Strassen and Coppersmith-Winograd algorithms are so large that they are faster than Algorithm 4.3.19 only for very large matrices.

Certain growth functions occur so often that they are given special names, as shown in Table 4.3.3. The functions in Table 4.3.3, with the exception of  $\Theta(n^k)$ , are arranged so that if  $\Theta(f(n))$  is above  $\Theta(g(n))$ , then  $f(n) \leq g(n)$  for all but finitely many positive integers  $n$ . Thus, if algorithms  $A$  and  $B$  have run times that are  $\Theta(f(n))$  and  $\Theta(g(n))$ , respectively, and  $\Theta(f(n))$  is above  $\Theta(g(n))$  in Table 4.3.3, then algorithm  $A$  is more time-efficient than algorithm  $B$  for sufficiently large inputs.

It is important to develop some feeling for the relative sizes of the functions in Table 4.3.3. In Figure 4.3.1 we have graphed some of these functions. Another way to develop some appreciation for the relative sizes of the functions  $f(n)$  in Table 4.3.3 is to determine how long it would take an algorithm to terminate whose run time is exactly  $f(n)$ . For this purpose, let us assume that we have a computer that can execute one step in 1 microsecond ( $10^{-6}$  sec). Table 4.3.1 shows the execution times, under this assumption, for various input sizes. Notice that it is practical to implement an algorithm that requires  $2^n$  steps for an input of size  $n$  only for very small input sizes. Algorithms requiring  $n^2$  or  $n^3$  steps also become impractical to implement, but for relatively larger input sizes. Also, notice the dramatic improvement that results when we move from  $n^2$  steps to  $n \lg n$  steps.

A problem that has a worst-case polynomial-time algorithm is considered to have a “good” algorithm; the interpretation is that such a problem has an efficient solution. Such problems are called **feasible** or **tractable**. Of course, if the worst-case time to solve the problem is proportional to a high-degree polynomial, the problem can still take a long time to solve. Fortunately, in many important cases, the polynomial bound has small degree.

**TABLE 4.3.3 ■ Common Growth Functions**

| Theta Form              | Name        |
|-------------------------|-------------|
| $\Theta(1)$             | Constant    |
| $\Theta(\lg \lg n)$     | Log log     |
| $\Theta(\lg n)$         | Log         |
| $\Theta(n)$             | Linear      |
| $\Theta(n \lg n)$       | $n \log n$  |
| $\Theta(n^2)$           | Quadratic   |
| $\Theta(n^3)$           | Cubic       |
| $\Theta(n^k), k \geq 1$ | Polynomial  |
| $\Theta(c^n), c > 1$    | Exponential |
| $\Theta(n!)$            | Factorial   |

A problem that does not have a worst-case polynomial-time algorithm is said to be **intractable**. Any algorithm, if there is one, that solves an intractable problem is guaranteed to take a long time to execute in the worst case, even for modest sizes of the input.

Certain problems are so hard that they have no algorithms at all. A problem for which there is no algorithm is said to be **unsolvable**. A large number of problems are known to be unsolvable, some of considerable practical importance. One of the earliest problems to be proved unsolvable is the **halting problem**: Given an arbitrary program and a set of inputs, will the program eventually halt?

A large number of solvable problems have an as yet undetermined status; they are thought to be intractable, but none of them has been proved to be intractable. (Most of these problems belong to the class of NP-complete problems; see [Johnsonbaugh] for details.) An example of an NP-complete problem is:

Given a collection  $\mathcal{C}$  of finite sets and a positive integer  $k \leq |\mathcal{C}|$ , does  $\mathcal{C}$  contain at least  $k$  mutually disjoint sets?

Other NP-complete problems include the traveling-salesperson problem and the Hamiltonian-cycle problem (see Section 8.3).

NP-complete problems have efficient (i.e., polynomial-time) algorithms to check whether a proposed solution is, in fact, a solution. For example, given a collection  $\mathcal{C}$  of finite sets and  $k$  sets in  $\mathcal{C}$ , it is easy and fast to check whether the  $k$  sets are mutually disjoint. (Just check each pair of sets!) On the other hand, no NP-complete problem is known to have an efficient algorithm. For example, given a collection  $\mathcal{C}$  of finite sets, *finding*  $k$  mutually disjoint sets in  $\mathcal{C}$  is, in general, difficult and time consuming. NP-complete problems also have the property that if any one of them has a polynomial-time algorithm, then *all* NP-complete problems have polynomial-time algorithms.

### 4.3 Problem-Solving Tips

- To derive a big oh notation for an expression  $f(n)$  directly, you must find a constant  $C_1$  and a simple expression  $g(n)$  (e.g.,  $n$ ,  $n \lg n$ ,  $n^2$ ) such that  $|f(n)| \leq C_1|g(n)|$  for all but finitely many  $n$ . Remember you're trying to derive an *inequality*, *not an equality*, so you can replace terms in  $f(n)$  with other terms if the result is *larger* (see, e.g., Example 4.3.3).
- To derive an omega notation for an expression  $f(n)$  directly, you must find a constant  $C_2$  and a simple expression  $g(n)$  such that  $|f(n)| \geq C_2|g(n)|$  for all but finitely many  $n$ . Again, you're trying to derive an *inequality* so you can replace terms in  $f(n)$  with other terms if the result is *smaller* (again, see Example 4.3.3).
- To derive a theta notation, you must derive both big oh and omega notations.
- Another way to derive big oh, omega, and theta estimates is to use known results:

| Expression                                        | Name                                         | Estimate          | Reference     |
|---------------------------------------------------|----------------------------------------------|-------------------|---------------|
| $a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$ | Polynomial                                   | $\Theta(n^k)$     | Theorem 4.3.4 |
| $1 + 2 + \dots + n$                               | Arithmetic Sum (Case $k = 1$ for Next Entry) | $\Theta(n^2)$     | Example 4.3.7 |
| $1^k + 2^k + \dots + n^k$                         | Sum of Powers                                | $\Theta(n^{k+1})$ | Example 4.3.8 |
| $\lg n!$                                          | $\log n$ Factorial                           | $\Theta(n \lg n)$ | Example 4.3.9 |

- To derive an asymptotic estimate for the time of an algorithm, count the number of steps  $t(n)$  required by the algorithm, and then derive an estimate for  $t(n)$  as described previously. Algorithms typically contain loops, in which case, deriving  $t(n)$  requires counting the number of iterations of the loops.

### 4.3 Review Exercises

- To what does “analysis of algorithms” refer?
- What is the worst-case time of an algorithm?
- What is the best-case time of an algorithm?
- What is the average-case time of an algorithm?
- Define  $f(n) = O(g(n))$ . What is this notation called?
- Give an intuitive interpretation of how  $f$  and  $g$  are related if  $f(n) = O(g(n))$ .
- Define  $f(n) = \Omega(g(n))$ . What is this notation called?
- Give an intuitive interpretation of how  $f$  and  $g$  are related if  $f(n) = \Omega(g(n))$ .
- Define  $f(n) = \Theta(g(n))$ . What is this notation called?
- Give an intuitive interpretation of how  $f$  and  $g$  are related if  $f(n) = \Theta(g(n))$ .

### 4.3 Exercises

Select a theta notation from Table 4.3.3 for each expression in Exercises 1–13.

1.  $6n + 1$
2.  $2n^2 + 1$
3.  $6n^3 + 12n^2 + 1$
4.  $3n^2 + 2n \lg n$
5.  $2 \lg n + 4n + 3n \lg n$
6.  $6n^6 + n + 4$
7.  $2 + 4 + 6 + \dots + 2n$
8.  $(6n + 1)^2$
9.  $(6n + 4)(1 + \lg n)$
10.  $\frac{(n+1)(n+3)}{n+2}$
11.  $\frac{(n^2 + \lg n)(n+1)}{n+n^2}$
12.  $2 + 4 + 8 + 16 + \dots + 2^n$
13.  $\lg[(2n)!]$

In Exercises 14–16, select a theta notation for  $f(n) + g(n)$ .

14.  $f(n) = \Theta(1)$ ,  $g(n) = \Theta(n^2)$
15.  $f(n) = 6n^3 + 2n^2 + 4$ ,  $g(n) = \Theta(n \lg n)$
16.  $f(n) = \Theta(n^{3/2})$ ,  $g(n) = \Theta(n^{5/2})$

In Exercises 17–26, select a theta notation from among

$$\begin{aligned} &\Theta(1), \quad \Theta(\lg n), \quad \Theta(n), \quad \Theta(n \lg n), \\ &\Theta(n^2), \quad \Theta(n^3), \quad \Theta(2^n), \quad \text{or} \quad \Theta(n!) \end{aligned}$$

for the number of times the statement  $x = x + 1$  is executed.

17. for  $i = 1$  to  $2n$   
 $x = x + 1$
18.  $i = 1$   
while ( $i \leq 2n$ ) {  
 $x = x + 1$   
 $i = i + 2$   
}
19. for  $i = 1$  to  $n$   
for  $j = 1$  to  $n$   
 $x = x + 1$

20. for  $i = 1$  to  $2n$   
for  $j = 1$  to  $n$   
 $x = x + 1$
21. for  $i = 1$  to  $n$   
for  $j = 1$  to  $\lfloor i/2 \rfloor$   
 $x = x + 1$
22. for  $i = 1$  to  $n$   
for  $j = 1$  to  $n$   
for  $k = 1$  to  $n$   
 $x = x + 1$
23. for  $i = 1$  to  $n$   
for  $j = 1$  to  $n$   
for  $k = 1$  to  $i$   
 $x = x + 1$
24. for  $i = 1$  to  $n$   
for  $j = 1$  to  $i$   
for  $k = 1$  to  $j$   
 $x = x + 1$
25.  $j = n$   
while ( $j \geq 1$ ) {  
  for  $i = 1$  to  $j$   
     $x = x + 1$   
   $j = \lfloor j/3 \rfloor$   
}
26.  $i = n$   
while ( $i \geq 1$ ) {  
  for  $j = 1$  to  $n$   
     $x = x + 1$   
   $i = \lfloor i/2 \rfloor$   
}
27. Find a theta notation for the number of times the statement  $x = x + 1$  is executed.  
  
 $i = 2$   
while ( $i < n$ ) {  
   $i = i^2$   
   $x = x + 1$   
}
28. Let  $t(n)$  be the total number of times that  $i$  is incremented and  $j$  is decremented in the following pseudocode, where  $a_1, a_2, \dots$  is a sequence of real numbers.

```

i = 1
j = n
while (i < j) {
 while (i < j ∧ ai < 0)
 i = i + 1
 while (i < j ∧ aj ≥ 0)
 j = j - 1
 if (i < j)
 swap(ai, aj)
}

```

Find a theta notation for  $t(n)$ .

- 29.** Find a theta notation for the worst-case time required by the following algorithm:

```

iskey(s, n, key) {
 for i = 1 to n - 1
 for j = i + 1 to n
 if (si + sj == key)
 return 1
 else
 return 0
}

```

- 30.** In addition to finding a theta notation in Exercises 1–29, prove that it is correct.
- 31.** Find the exact number of comparisons (lines 10, 15, 17, 24, and 26) required by the following algorithm when  $n$  is even and when  $n$  is odd. Find a theta notation for this algorithm.

Input:  $s_1, s_2, \dots, s_n, n$

Output:  $large$  (the largest item in  $s_1, s_2, \dots, s_n$ ),  
 $small$  (the smallest item in  $s_1, s_2, \dots, s_n$ )

```

1. large_small(s, n, large, small) {
2. if (n == 1) {
3. large = s1
4. small = s1
5. return
6. }
7. m = 2 $\lfloor n/2 \rfloor
8. i = 1
9. while (i ≤ m - 1) {
10. if (si > si+1)
11. swap(si, si+1)
12. i = i + 2
13. }
14. if (n > m) {
15. if (sm-1 > sn)
16. swap(sm-1, sn)
17. if (sn > sm)
18. swap(sm, sn)
19. }
20. small = s1
21. large = s2
22. i = 3
23. while (i ≤ m - 1) {$
```

```

24. if (si < small)
25. small = si
26. if (si+1 > large)
27. large = si+1
28. i = i + 2
29. }
30. }

```

- 32.** This exercise shows another way to guess a formula for  $1 + 2 + \dots + n$ .

Example 4.3.7 suggests that

$$1 + 2 + \dots + n = An^2 + Bn + C \quad \text{for all } n,$$

for some constants  $A$ ,  $B$ , and  $C$ . Assuming that this is true, plug in  $n = 1, 2, 3$  to obtain three equations in the three unknowns  $A$ ,  $B$ , and  $C$ . Now solve for  $A$ ,  $B$ , and  $C$ . The resulting formula can now be proved using mathematical induction (see Section 2.4).

- 33.** Suppose that  $a > 1$  and that  $f(n) = \Theta(\log_a n)$ . Show that  $f(n) = \Theta(\lg n)$ .
- 34.** Show that  $n! = O(n^n)$ .
- 35.** Show that  $2^n = O(n!)$ .
- 36.** By using an argument like the one shown in Examples 4.3.7–4.3.9 or otherwise, prove that  $\sum_{i=1}^n i \lg i = \Theta(n^2 \lg n)$ .
- \*37.** Show that  $n^{n+1} = O(2^{n^2})$ .
- 38.** Show that  $\lg(n^k + c) = \Theta(\lg n)$  for every fixed  $k > 0$  and  $c > 0$ .
- 39.** Show that if  $n$  is a power of 2, say  $n = 2^k$ , then

$$\sum_{i=0}^k \lg(n/2^i) = \Theta(\lg^2 n).$$

- 40.** Suppose that  $f(n) = O(g(n))$ , and  $f(n) \geq 0$  and  $g(n) > 0$  for all  $n \geq 1$ . Show that for some constant  $C$ ,  $f(n) \leq Cg(n)$  for all  $n \geq 1$ .
- 41.** State and prove a result for  $\Omega$  similar to that for Exercise 40.
- 42.** State and prove a result for  $\Theta$  similar to that for Exercises 40 and 41.

Determine whether each statement in Exercises 43–68 is true or false. If the statement is true, prove it. If the statement is false, give a counterexample. Assume that the functions  $f$ ,  $g$ , and  $h$  take on only positive values.

- 43.**  $n^n = O(2^n)$
- 44.**  $2 + \sin n = O(2 + \cos n)$
- 45.**  $(2n)^2 = O(n^2)$
- 46.**  $(2n)^2 = \Omega(n^2)$
- 47.**  $(2n)^2 = \Theta(n^2)$
- 48.**  $2^{2n} = O(2^n)$
- 49.**  $2^{2n} = \Omega(2^n)$

50.  $2^{2n} = \Theta(2^n)$   
 51.  $n! = O((n+1)!)$   
 52.  $n! = \Omega((n+1)!)$   
 53.  $n! = \Theta((n+1)!)$   
 54.  $\lg(2n)^2 = O(\lg n^2)$   
 55.  $\lg(2n)^2 = \Omega(\lg n^2)$   
 56.  $\lg(2n)^2 = \Theta(\lg n^2)$   
 57.  $\lg 2^{2n} = O(\lg 2^n)$   
 58.  $\lg 2^{2n} = \Omega(\lg 2^n)$   
 59.  $\lg 2^{2n} = \Theta(\lg 2^n)$

60. If  $f(n) = \Theta(h(n))$  and  $g(n) = \Theta(h(n))$ , then  $f(n) + g(n) = \Theta(h(n))$ .

61. If  $f(n) = \Theta(g(n))$ , then  $cf(n) = \Theta(g(n))$  for any  $c \neq 0$ .

62. If  $f(n) = \Theta(g(n))$ , then  $2^{f(n)} = \Theta(2^{g(n)})$ .

63. If  $f(n) = \Theta(g(n))$ , then  $\lg f(n) = \Theta(\lg g(n))$ . Assume that  $f(n) \geq 1$  and  $g(n) \geq 1$  for all  $n = 1, 2, \dots$ .

64. If  $f(n) = O(g(n))$ , then  $g(n) = O(f(n))$ .

65. If  $f(n) = O(g(n))$ , then  $g(n) = \Omega(f(n))$ .

66. If  $f(n) = \Theta(g(n))$ , then  $g(n) = \Theta(f(n))$ .

67.  $f(n) + g(n) = \Theta(h(n))$ , where  $h(n) = \max\{f(n), g(n)\}$

68.  $f(n) + g(n) = \Theta(h(n))$ , where  $h(n) = \min\{f(n), g(n)\}$

69. Write out exactly what  $f(n) \neq O(g(n))$  means.

70. What is wrong with the following argument that purports to show that we cannot simultaneously have  $f(n) \neq O(g(n))$  and  $g(n) \neq O(f(n))$ ?

If  $f(n) \neq O(g(n))$ , then for every  $C > 0$ ,  $|f(n)| > C|g(n)|$ . In particular,  $|f(n)| > 2|g(n)|$ . If  $g(n) \neq O(f(n))$ , then for every  $C > 0$ ,  $|g(n)| > C|f(n)|$ . In particular,  $|g(n)| > 2|f(n)|$ . But now

$$|f(n)| > 2|g(n)| > 4|f(n)|.$$

Cancelling  $|f(n)|$  gives  $1 > 4$ , which is a contradiction. Therefore, we cannot simultaneously have  $f(n) \neq O(g(n))$  and  $g(n) \neq O(f(n))$ .

★71. Find functions  $f$  and  $g$  satisfying

$$f(n) \neq O(g(n)) \quad \text{and} \quad g(n) \neq O(f(n)).$$

★72. Give an example of increasing positive functions  $f$  and  $g$  defined on the positive integers for which

$$f(n) \neq O(g(n)) \quad \text{and} \quad g(n) \neq O(f(n)).$$

★73. Prove that  $n^k = O(c^n)$  for all  $k = 1, 2, \dots$  and  $c > 1$ .

74. Find functions  $f, g, h$ , and  $t$  satisfying

$$f(n) = \Theta(g(n)), \quad h(n) = \Theta(t(n)), \\ f(n) - h(n) \neq \Theta(g(n) - t(n)).$$

75. Suppose that the worst-case time of an algorithm is  $\Theta(n)$ . What is the error in the following reasoning? Since  $2n = \Theta(n)$ , the worst-case time to run the algorithm with input of

size  $2n$  will be approximately the same as the worst-case time to run the algorithm with input of size  $n$ .

76. Does  $f(n) = O(g(n))$  define an equivalence relation on the set of real-valued functions on  $\{1, 2, \dots\}$ ?

77. Does  $f(n) = \Theta(g(n))$  define an equivalence relation on the set of real-valued functions on  $\{1, 2, \dots\}$ ?

78. [Requires the integral]

(a) Show, by consulting the figure, that

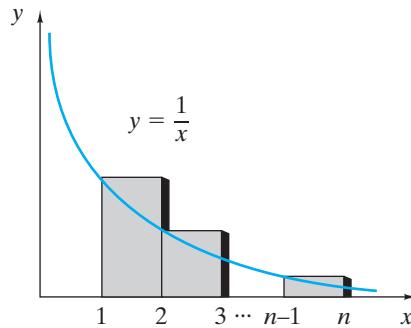
$$\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} < \log_e n.$$

(b) Show, by consulting the figure, that

$$\log_e n < 1 + \frac{1}{2} + \cdots + \frac{1}{n-1}.$$

(c) Use parts (a) and (b) to show that

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



79. [Requires the integral] Use an argument like the one shown in Exercise 78 to show that

$$\frac{n^{m+1}}{m+1} < 1^m + 2^m + \cdots + n^m < \frac{(n+1)^{m+1}}{m+1},$$

where  $m$  is a positive integer.

80. By using the formula

$$\frac{b^{n+1} - a^{n+1}}{b - a} = \sum_{i=0}^n a^i b^{n-i} \quad 0 \leq a < b$$

or otherwise, prove that

$$\frac{b^{n+1} - a^{n+1}}{b - a} < (n+1)b^n \quad 0 \leq a < b.$$

81. Take  $a = 1 + 1/(n+1)$  and  $b = 1 + 1/n$  in the inequality of Exercise 80 to prove that the sequence  $\{(1 + 1/n)^n\}$  is increasing.

82. Take  $a = 1$  and  $b = 1 + 1/(2n)$  in the inequality of Exercise 80 to prove that

$$\left(1 + \frac{1}{2n}\right)^n < 2$$

for all  $n \geq 1$ . Use the preceding exercise to conclude that

$$\left(1 + \frac{1}{n}\right)^n < 4$$

for all  $n \geq 1$ .

The method used to prove the results of this exercise and its predecessor is apparently due to Fort in 1862 (see [Chrystal, vol. II, page 77]).

- 83.** By using the preceding two exercises or otherwise, prove that

$$\frac{1}{n} \leq \lg(n+1) - \lg n < \frac{2}{n}$$

for all  $n \geq 1$ .

- 84.** Use the preceding exercise to prove that

$$\sum_{i=1}^n \frac{1}{i} = \Theta(\lg n).$$

(Compare with Exercise 78.)

- 85.** Prove that the sequence  $\{n^{1/n}\}_{n=3}^\infty$  is decreasing.

- 86.** Prove that if  $0 \leq a < b$ , then

$$\frac{b^{n+1} - a^{n+1}}{b - a} > (n+1)a^n.$$

- 87.** Find appropriate values for  $a$  and  $b$  in the inequality in the preceding exercise to prove that the sequence  $\{(1 - 1/n)^n\}_{n=1}^\infty$  is increasing and bounded above by  $4/9$ .

- 88.** By using the result of the preceding exercise, or otherwise, prove that the sequence  $\{(1 + 1/n)^{n+1}\}_{n=1}^\infty$  is decreasing.

- 89.** By using the result of the preceding exercise, or otherwise, prove that

$$\lg(n+1) - \lg n \leq \frac{2}{n+1}$$

for all  $n \geq 1$ .

- 90.** What is wrong with the following “proof” that any algorithm has a run time that is  $O(n)$ ?

We must show that the time required for an input of size  $n$  is at most a constant times  $n$ .

### Basis Step

Suppose that  $n = 1$ . If the algorithm takes  $C$  units of time for an input of size 1, the algorithm takes at most  $C \cdot 1$  units of time. Thus the assertion is true for  $n = 1$ .

### Inductive Step

Assume that the time required for an input of size  $n$  is at most  $C'n$  and that the time for processing an additional item is  $C''$ . Let  $C$  be the maximum of  $C'$  and  $C''$ . Then the total time required for an input of size  $n+1$  is at most

$$C'n + C'' \leq Cn + C = C(n+1).$$

The Inductive Step has been verified.

By induction, for input of size  $n$ , the time required is at most a constant time  $n$ . Therefore, the run time is  $O(n)$ .

In Exercises 91–96, determine whether the statement is true or false. If the statement is true, prove it. If the statement is false, give a counterexample. Assume that  $f$  and  $g$  are real-valued functions defined on the set of positive integers and that  $g(n) \neq 0$  for  $n \geq 1$ . These exercises require calculus.

- 91.** If

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0,$$

then  $f(n) = O(g(n))$ .

- 92.** If

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0,$$

then  $f(n) = \Theta(g(n))$ .

- 93.** If

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c \neq 0,$$

then  $f(n) = O(g(n))$ .

- 94.** If

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c \neq 0,$$

then  $f(n) = \Theta(g(n))$ .

- 95.** If  $f(n) = O(g(n))$ , then

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$$

exists and is equal to some real number.

- 96.** If  $f(n) = \Theta(g(n))$ , then

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$$

exists and is equal to some real number.

- \*97.** Use induction to prove that

$$\lg n! \geq \frac{n}{2} \lg \frac{n}{2}.$$

- 98.** [Requires calculus] Let  $\ln x$  denote the natural logarithm ( $\log_e x$ ) of  $x$ . Use the integral to obtain the estimate

$$n \ln n - n \leq \sum_{k=1}^n \ln k = \ln n!, \quad n \geq 1.$$

- 99.** Use the result of Exercise 98 and the change-of-base formula for logarithms to obtain the formula

$$n \lg n - n \lg e \leq \lg n!, \quad n \geq 1.$$

- 100.** Deduce

$$\lg n! \geq \frac{n}{2} \lg \frac{n}{2}$$

from the inequality of Exercise 99.

**Problem-Solving Corner****Design and Analysis of an Algorithm****Problem**

Develop and analyze an algorithm that returns the maximum sum of consecutive values in the numerical sequence  $s_1, \dots, s_n$ . In mathematical notation, the problem is to find the maximum sum of the form  $s_j + s_{j+1} + \dots + s_i$ . *Example:* If the sequence is

27 6 -50 21 -3 14 16 -8 42 33 -21 9,

the algorithm returns 115—the sum of

21 -3 14 16 -8 42 33.

If all the numbers in a sequence are negative, the maximum sum of consecutive values is defined to be 0. (The idea is that the maximum of 0 is achieved by taking an “empty” sum.)

**Attacking the Problem**

In developing an algorithm, a good way to start is to ask the question, “How would I solve this problem by hand?” At least initially, take a straightforward approach. Here we might just list the sums of *all* consecutive values and pick the largest. For the example sequence, the sums are as follows:

| i  | j   |     |     |     |    |    |    |    |    |    |     |    |
|----|-----|-----|-----|-----|----|----|----|----|----|----|-----|----|
|    | 1   | 2   | 3   | 4   | 5  | 6  | 7  | 8  | 9  | 10 | 11  | 12 |
| 1  | 27  |     |     |     |    |    |    |    |    |    |     |    |
| 2  | 33  | 6   |     |     |    |    |    |    |    |    |     |    |
| 3  | -17 | -44 | -50 |     |    |    |    |    |    |    |     |    |
| 4  | 4   | -23 | -29 | 21  |    |    |    |    |    |    |     |    |
| 5  | 1   | -26 | -32 | 18  | -3 |    |    |    |    |    |     |    |
| 6  | 15  | -12 | -18 | 32  | 11 | 14 |    |    |    |    |     |    |
| 7  | 31  | 4   | -2  | 48  | 27 | 30 | 16 |    |    |    |     |    |
| 8  | 23  | -4  | -10 | 40  | 19 | 22 | 8  | -8 |    |    |     |    |
| 9  | 65  | 38  | 32  | 82  | 61 | 64 | 50 | 34 | 42 |    |     |    |
| 10 | 98  | 71  | 65  | 115 | 94 | 97 | 83 | 67 | 75 | 33 |     |    |
| 11 | 77  | 50  | 44  | 94  | 73 | 76 | 62 | 46 | 54 | 12 | -21 |    |
| 12 | 86  | 59  | 53  | 103 | 82 | 85 | 71 | 55 | 63 | 21 | -12 | 9  |

The entry in column  $j$ , row  $i$ , is the sum  $s_j + \dots + s_i$ . For example, the entry in column 4, row 7, is 48—the sum

$$s_4 + s_5 + s_6 + s_7 = 21 + -3 + 14 + 16 = 48.$$

By inspection, we find that 115 is the largest sum.

**Finding a Solution**

We begin by writing pseudocode for the straightforward algorithm that computes all consecutive sums and finds the largest:

Input:  $s_1, \dots, s_n$

Output:  $\max$

*max\_sum1*( $s, n$ ) {

//  $sum_{ji}$  is the sum  $s_j + \dots + s_i$ .  
for  $i = 1$  to  $n$  {  
    for  $j = 1$  to  $i - 1$   
         $sum_{ji} = sum_{j,i-1} + s_i$   
         $sum_{ii} = s_i$   
    }

// step through  $sum_{ji}$  and find the maximum

$max = 0$

for  $i = 1$  to  $n$

    for  $j = 1$  to  $i$   
        if ( $sum_{ji} > max$ )  
             $max = sum_{ji}$

return  $max$

}

The first nested for loops compute the sums  
 $sum_{ji} = s_j + \dots + s_i$ .  
The computation relies on the fact that  

$$\begin{aligned} sum_{ji} &= s_j + \dots + s_i = s_j + \dots + s_{i-1} + s_i \\ &= sum_{j,i-1} + s_i. \end{aligned}$$

The second nested for loops step through  $sum_{ji}$  and find the largest value.

Since each of the nested for loops takes time  $\Theta(n^2)$ ,  $max\_sum1$ 's time is  $\Theta(n^2)$ .

We can improve the actual time, but not the asymptotic time, of the algorithm by computing the maximum within the same nested for loops in which we compute  $sum_{ji}$ :

Input:  $s_1, \dots, s_n$

Output:  $max$

```
max_sum2(s, n) {
 // sumji is the sum $s_j + \dots + s_i$.
 max = 0
 for i = 1 to n {
 for j = 1 to i - 1 {
 sumji = sumj,i-1 + si
 if (sumji > max)
 max = sumji
 }
 sumii = si
 if (sumii > max)
 max = sumii
 }
 return max
}
```

Since the nested for loops take time  $\Theta(n^2)$ ,  $max\_sum2$ 's time is  $\Theta(n^2)$ . To reduce the asymptotic time, we need to take a hard look at the pseudocode to see where it can be improved.

Two key observations lead to improved time. First, since we are looking only for the *maximum* sum, there is no need to record all of the sums; we will store only the maximum sum that ends at index  $i$ . Second, the line

$$sum_{ji} = sum_{j,i-1} + s_i$$

shows how a consecutive sum that ends at index  $i - 1$  is related to a consecutive sum that ends at index  $i$ . The maximum can be computed by using a similar formula. If  $sum$  is the maximum consecutive sum that ends at index  $i - 1$ , the maximum consecutive sum that ends at index  $i$  is obtained by adding  $s_i$  to  $sum$  provided that  $sum + s_i$  is positive. (If some sum of consecutive terms that ends at index  $i$  exceeds  $sum + s_i$ , we could remove  $s_i$  and obtain a sum of consecutive terms ending at index  $i - 1$  that exceeds  $sum$ , which is impossible.) If  $sum + s_i \leq 0$ , the maximum consecutive sum that ends at index  $i$  is obtained by taking no terms and has value 0. Thus we may compute the maximum consecutive sum that ends at index  $i$  by executing

```
if (sum + si > 0)
 sum = sum + si
else
 sum = 0
```

## Formal Solution

Input:  $s_1, \dots, s_n$

Output:  $max$

```
max_sum3(s, n) {
 // max is the maximum sum seen so far.
 // After the i th iteration of the for
 // loop, sum is the largest consecutive
 // sum that ends at index i .
 max = 0
 sum = 0
 for i = 1 to n {
 if (sum + si > 0)
 sum = sum + si
 else
 sum = 0
 if (sum > max)
 max = sum
 }
 return max
}
```

Since this algorithm has a single for loop that runs from 1 to  $n$ ,  $max\_sum3$ 's time is  $\Theta(n)$ . The asymptotic time of this algorithm cannot be further improved. To find the maximum sum of consecutive values, we must at least look at each element in the sequence, which takes time  $\Theta(n)$ .

## Summary of Problem-Solving Techniques

- In developing an algorithm, a good way to start is to ask the question, “How would I solve this problem by hand?”
- In developing an algorithm, initially take a straightforward approach.
- After developing an algorithm, take a close look at the pseudocode to see where it can be improved. Look at the parts that perform key computations to gain insight into how to enhance the algorithm’s efficiency.
- As in mathematical induction, extend a solution of a smaller problem to a larger problem. (In this problem, we extended a sum that ends at index  $i - 1$  to a sum that ends at index  $i$ .)

- Don't repeat computations. (In this problem, we extended a sum that ends at index  $i - 1$  to a sum that ends at index  $i$  by adding an additional term rather than by computing the sum that ends at index  $i$  from scratch. This latter method would have meant recomputing the sum that ends at index  $i - 1$ .)

### Comments

According to [Bentley], the problem discussed in this section is the one-dimensional version of the original two-dimensional problem that dealt with pattern

matching in digital images. The original problem was to find the maximum sum in a rectangular submatrix of an  $n \times n$  matrix of real numbers.

### Exercises

- Modify *max\_sum3* so that it computes not only the maximum sum of consecutive values but also the indexes of the first and last terms of a maximum-sum subsequence. If there is no maximum-sum subsequence (which would happen, for example, if all of the values of the sequence were negative), the algorithm should set the first and last indexes to zero.

## 4.4

## Recursive Algorithms

### Go Online

For more on recursion, see [goo.gl/ZwpPlu](http://goo.gl/ZwpPlu)

A **recursive function** (pseudocode) is a function that invokes itself. A **recursive algorithm** is an algorithm that contains a recursive function. Recursion is a powerful, elegant, and natural way to solve a large class of problems. A problem in this class can be solved using a *divide-and-conquer* technique in which the problem is decomposed into problems of the same type as the original problem. Each subproblem, in turn, is decomposed further until the process yields subproblems that can be solved in a straightforward way. Finally, solutions to the subproblems are combined to obtain a solution to the original problem.

### Example 4.4.1

**TABLE 4.4.1** ■ Decomposing the Factorial Problem

| Problem | Simplified Problem |
|---------|--------------------|
| 5!      | 5 · 4!             |
| 4!      | 4 · 3!             |
| 3!      | 3 · 2!             |
| 2!      | 2 · 1!             |
| 1!      | 1 · 0!             |
| 0!      | None               |

**TABLE 4.4.2** ■ Combining Subproblems of the Factorial Problem

| Problem | Solution              |
|---------|-----------------------|
| 0!      | 1                     |
| 1!      | 1 · 0! = 1            |
| 2!      | 2 · 1! = 2            |
| 3!      | 3 · 2! = 3 · 2 = 6    |
| 4!      | 4 · 3! = 4 · 6 = 24   |
| 5!      | 5 · 4! = 5 · 24 = 120 |

Recall that if  $n \geq 1$ ,  $n! = n(n - 1) \cdots 2 \cdot 1$ , and  $0! = 1$ . Notice that if  $n \geq 2$ ,  $n$  factorial can be written “in terms of itself” since, if we “peel off”  $n$ , the remaining product is simply  $(n - 1)!$ ; that is,

$$n! = n(n - 1)(n - 2) \cdots 2 \cdot 1 = n \cdot (n - 1)!.$$

For example,

$$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 5 \cdot 4!.$$

The equation

$$n! = n \cdot (n - 1)!, \quad (4.4.1)$$

which happens to be true even when  $n = 1$ , shows how to decompose the original problem (compute  $n!$ ) into increasingly simpler subproblems [compute  $(n - 1)!$ , compute  $(n - 2)!$ , ...] until the process reaches the straightforward problem of computing  $0!$ . The solutions to these subproblems can then be combined, by multiplying, to solve the original problem.

For example, the problem of computing  $5!$  is reduced to computing  $4!$ ; the problem of computing  $4!$  is reduced to computing  $3!$ ; and so on. Table 4.4.1 summarizes this process.

Once the problem of computing  $5!$  has been reduced to solving subproblems, the solution to the simplest subproblem can be used to solve the next simplest subproblem, and so on, until the original problem has been solved. Table 4.4.2 shows how the subproblems are combined to compute  $5!$ .

Next, we write a recursive algorithm that computes factorials. The algorithm is a direct translation of equation (4.4.1).

**Algorithm 4.4.2****Computing  $n$  Factorial**

This recursive algorithm computes  $n!$ .

Input:  $n$ , an integer greater than or equal to 0  
 Output:  $n!$

```

1. factorial(n) {
2. if ($n == 0$)
3. return 1
4. return $n * factorial(n - 1)$
5. }
```

We show how Algorithm 4.4.2 computes  $n!$  for several values of  $n$ . If  $n = 0$ , at line 3 the function correctly returns the value 1.

If  $n = 1$ , we proceed to line 4 since  $n \neq 0$ . We use this function to compute  $0!$ . We have just observed that the function computes 1 as the value of  $0!$ . At line 4, the function correctly computes the value of  $1!:$

$$n \cdot (n - 1)! = 1 \cdot 0! = 1 \cdot 1 = 1.$$

If  $n = 2$ , we proceed to line 4 since  $n \neq 0$ . We use this function to compute  $1!$ . We have just observed that the function computes 1 as the value of  $1!$ . At line 4, the function correctly computes the value of  $2!:$

$$n \cdot (n - 1)! = 2 \cdot 1! = 2 \cdot 1 = 2.$$

If  $n = 3$  we proceed to line 4 since  $n \neq 0$ . We use this function to compute  $2!$ . We have just observed that the function computes 2 as the value of  $2!$ . At line 4, the function correctly computes the value of  $3!:$

$$n \cdot (n - 1)! = 3 \cdot 2! = 3 \cdot 2 = 6.$$

The preceding arguments may be generalized using mathematical induction to *prove* that Algorithm 4.4.2 correctly returns the value of  $n!$  for any nonnegative integer  $n$ .

**Theorem 4.4.3**

Algorithm 4.4.2 returns the value of  $n!$ ,  $n \geq 0$ .

**Proof****Basis Step ( $n = 0$ )**

We have already observed that if  $n = 0$ , Algorithm 4.4.2 correctly returns the value of  $0!$  (1).

**Inductive Step**

Assume that Algorithm 4.4.2 correctly returns the value of  $(n - 1)!$ ,  $n > 0$ . Now suppose that  $n$  is input to Algorithm 4.4.2. Since  $n \neq 0$ , when we execute the function in Algorithm 4.4.2 we proceed to line 4. By the inductive assumption, the function correctly computes the value of  $(n - 1)!$ . At line 4, the function correctly computes the value  $(n - 1)! \cdot n = n!$ .

Therefore, Algorithm 4.4.2 correctly returns the value of  $n!$  for every integer  $n \geq 0$ . ◀

If executed by a computer, Algorithm 4.4.2 would typically not be as efficient as a nonrecursive version because of the overhead of the recursive calls.

There must be some situations in which a recursive function does *not* invoke itself; otherwise, it would invoke itself forever. In Algorithm 4.4.2, if  $n = 0$ , the function does not invoke itself. We call the values for which a recursive function does not invoke itself the *base cases*. To summarize, every recursive function must have base cases.

We have shown how mathematical induction may be used to prove that a recursive algorithm computes the value it claims to compute. The link between mathematical induction and recursive algorithms runs deep. Often a proof by mathematical induction can be considered to be an algorithm to compute a value or to carry out a particular construction. The Basis Step of a proof by mathematical induction corresponds to the base cases of a recursive function, and the Inductive Step of a proof by mathematical induction corresponds to the part of a recursive function where the function calls itself.

In Example 2.4.7, we gave a proof using mathematical induction that, given an  $n \times n$  deficient board (a board with one square removed), where  $n$  is a power of 2, we can tile the board with right trominoes (three squares that form an “L”; see Figure 2.4.4). We now translate the inductive proof into a recursive algorithm to construct a tiling by right trominoes of an  $n \times n$  deficient board where  $n$  is a power of 2.

### Algorithm 4.4.4

#### Go Online

For a C program implementing this algorithm, see [goo.gl/rZYmnK](http://goo.gl/rZYmnK)

#### Tiling a Deficient Board with Trominoes

This algorithm constructs a tiling by right trominoes of an  $n \times n$  deficient board where  $n$  is a power of 2.

**Input:**  $n$ , a power of 2 (the board size); and the location  $L$  of the missing square  
**Output:** A tiling of an  $n \times n$  deficient board

```

1. tile(n, L) {
2. if ($n == 2$) {
3. // the board is a right tromino T
4. tile with T
5. return
6. }
7. divide the board into four $(n/2) \times (n/2)$ boards
8. rotate the board so that the missing square is in the upper-left quadrant
9. place one right tromino in the center // as in Figure 2.4.5
10. // consider each of the squares covered by the center tromino as
11. // missing, and denote the missing squares as m_1, m_2, m_3, m_4
12. tile($n/2, m_1$)
13. tile($n/2, m_2$)
14. tile($n/2, m_3$)
15. tile($n/2, m_4$)
16. }
```

Using the method of the proof of Theorem 4.4.3, we can prove that Algorithm 4.4.4 is correct (see Exercise 4).

We present one final example of a recursive algorithm.

### Example 4.4.5

A robot can take steps of 1 meter or 2 meters. We write an algorithm to calculate the number of ways the robot can walk  $n$  meters. As examples:

| <i>Distance</i> | <i>Sequence of Steps</i>                               | <i>Number of Ways to Walk</i> |
|-----------------|--------------------------------------------------------|-------------------------------|
| 1               | 1                                                      | 1                             |
| 2               | 1, 1 or 2                                              | 2                             |
| 3               | 1, 1, 1 or 1, 2 or 2, 1                                | 3                             |
| 4               | 1, 1, 1, 1 or 1, 1, 2<br>or 1, 2, 1 or 2, 1, 1 or 2, 2 | 5                             |

Let  $\text{walk}(n)$  denote the number of ways the robot can walk  $n$  meters. We have observed that  $\text{walk}(1) = 1$  and  $\text{walk}(2) = 2$ . Now suppose that  $n > 2$ . The robot can begin by taking a step of 1 meter or a step of 2 meters. If the robot begins by taking a 1-meter step, a distance of  $n - 1$  meters remains; but, by definition, the remainder of the walk can be completed in  $\text{walk}(n - 1)$  ways. Similarly, if the robot begins by taking a 2-meter step, a distance of  $n - 2$  meters remains and, in this case, the remainder of the walk can be completed in  $\text{walk}(n - 2)$  ways. Since the walk must begin with either a 1-meter or a 2-meter step, all of the ways to walk  $n$  meters are accounted for. We obtain the formula

$$\text{walk}(n) = \text{walk}(n - 1) + \text{walk}(n - 2). \quad (4.4.2)$$

For example,

$$\text{walk}(4) = \text{walk}(3) + \text{walk}(2) = 3 + 2 = 5.$$

We can write a recursive algorithm to compute  $\text{walk}(n)$  by translating equation (4.4.2) directly into an algorithm. The base cases are  $n = 1$  and  $n = 2$ . 

### Algorithm 4.4.6

#### Robot Walking

This algorithm computes the function defined by

$$\text{walk}(n) = \begin{cases} 1, & n = 1 \\ 2, & n = 2 \\ \text{walk}(n - 1) + \text{walk}(n - 2) & n > 2. \end{cases}$$

Input:  $n$

Output:  $\text{walk}(n)$

```
walk(n) {
 if (n == 1 ∨ n == 2)
 return n
 return walk(n - 1) + walk(n - 2)
}
```

### Go Online

For a C program implementing this algorithm, see [goo.gl/m754WF](http://goo.gl/m754WF)

### Go Online

For more on the Fibonacci sequence, see [goo.gl/ZwpP1u](http://goo.gl/ZwpP1u)

Using the method of the proof of Theorem 4.4.3, we can prove that Algorithm 4.4.6 is correct (see Exercise 7).

The sequence  $\text{walk}(1), \text{walk}(2), \text{walk}(3), \dots$ , whose values begin 1, 2, 3, 5, 8, 13, ..., is related to the Fibonacci sequence. The **Fibonacci sequence**  $\{f_n\}$  is defined by the equations

$$f_1 = 1, \quad f_2 = 1, \quad f_n = f_{n-1} + f_{n-2} \quad n \geq 3.$$



**Figure 4.4.1** A pine cone. There are 13 clockwise spirals (marked with white thread) and 8 counterclockwise spirals (marked with dark thread).  
[Photo by the author; pine cone courtesy of André Berthiaume and Sigrid (Anne) Settle.]

The Fibonacci sequence begins

$$1, 1, 2, 3, 5, 8, 13, \dots$$

Since  $\text{walk}(1) = f_2$ ,  $\text{walk}(2) = f_3$ , and

$$\text{walk}(n) = \text{walk}(n - 1) + \text{walk}(n - 2), \quad f_n = f_{n-1} + f_{n-2} \quad \text{for all } n \geq 3,$$

it follows that

$$\text{walk}(n) = f_{n+1} \quad \text{for all } n \geq 1.$$

(The argument can be formalized using mathematical induction; see Exercise 8.)

The Fibonacci sequence is named in honor of Leonardo Fibonacci (ca. 1170–1250), an Italian merchant and mathematician. The sequence originally arose in a puzzle about rabbits (see Exercises 18 and 19). After returning from the Orient in 1202, Fibonacci wrote his most famous work, *Liber Abaci* (available in an English translation by [Sigler]), which, in addition to containing what we now call the Fibonacci sequence, advocated the use of Hindu-Arabic numerals. This book was one of the main influences in bringing the decimal number system to Western Europe. Fibonacci signed much of his work “Leonardo Bigollo.” *Bigollo* translates as “traveler” or “blockhead.” There is some evidence that Fibonacci enjoyed having his contemporaries consider him a blockhead for advocating the new number system.

The Fibonacci sequence pops up in unexpected places. Figure 4.4.1 shows a pine cone with 13 clockwise spirals and 8 counterclockwise spirals. Many plants distribute their seeds as evenly as possible, thus maximizing the space available for each seed. The pattern in which the number of spirals is a Fibonacci number provides the most even distribution (see [Naylor, Mitchison]). In Section 5.3, the Fibonacci sequence appears in the analysis of the Euclidean algorithm.

#### Example 4.4.7

Use mathematical induction to show that

$$\sum_{k=1}^n f_k = f_{n+2} - 1 \quad \text{for all } n \geq 1.$$

**SOLUTION** For the basis step ( $n = 1$ ), we must show that

$$\sum_{k=1}^1 f_k = f_3 - 1.$$

Since  $\sum_{k=1}^1 f_k = f_1 = 1$  and  $f_3 - 1 = 2 - 1 = 1$ , the equation is verified.  
For the inductive step, we assume case  $n$

$$\sum_{k=1}^n f_k = f_{n+2} - 1$$

and prove case  $n + 1$

$$\sum_{k=1}^{n+1} f_k = f_{n+3} - 1.$$

Now

$$\begin{aligned} \sum_{k=1}^{n+1} f_k &= \sum_{k=1}^n f_k + f_{n+1} \\ &= (f_{n+2} - 1) + f_{n+1} \quad \text{by the inductive assumption} \\ &= f_{n+1} + f_{n+2} - 1 \\ &= f_{n+3} - 1. \end{aligned}$$

The last equality is true because of the definition of the Fibonacci numbers:

$$f_n = f_{n-1} + f_{n-2} \quad \text{for all } n \geq 3.$$

Since the basis step and the inductive step have been verified, the given equation is true for all  $n \geq 1$ . 

#### 4.4 Problem-Solving Tips

A recursive function is a function that invokes itself. The key to writing a recursive function is to find a smaller instance of the problem within the larger problem. For example, we can compute  $n!$  recursively because  $n! = n \cdot (n - 1)!$  for all  $n \geq 1$ . The situation is analogous to the inductive step in mathematical induction when we must find a smaller case (e.g., case  $n$ ) within the larger case (e.g., case  $n + 1$ ).

As another example, tiling an  $n \times n$  deficient board with trominoes when  $n$  is a power of 2 can be done recursively because we can find four  $(n/2) \times (n/2)$  subboards within the original  $n \times n$  board. Note the similarity of the tiling algorithm to the inductive step of the proof that every  $n \times n$  deficient board can be tiled with trominoes when  $n$  is a power of 2.

To prove a statement about the Fibonacci numbers, use the equation

$$f_n = f_{n-1} + f_{n-2} \quad \text{for all } n \geq 3.$$

The proof will often use mathematical induction *and* the previous equation (see Example 4.4.7).

## 4.4 Review Exercises

1. What is a recursive algorithm?
2. What is a recursive function?
3. Give an example of a recursive function.
4. Explain how the divide-and-conquer technique works.
5. What is a base case in a recursive function?
6. Why must every recursive function have a base case?
7. How is the Fibonacci sequence defined?
8. Give the first four values of the Fibonacci sequence.

## 4.4 Exercises

1. Trace Algorithm 4.4.2 for  $n = 4$ .
2. Trace Algorithm 4.4.4 when  $n = 4$  and the missing square is the upper-left corner square.
3. Trace Algorithm 4.4.4 when  $n = 8$  and the missing square is four from the left and six from the top.
4. Prove that Algorithm 4.4.4 is correct.
5. Trace Algorithm 4.4.6 for  $n = 4$ .
6. Trace Algorithm 4.4.6 for  $n = 5$ .
7. Prove that Algorithm 4.4.6 is correct.
8. Prove that

$$\text{walk}(n) = f_{n+1} \quad \text{for all } n \geq 1.$$

9. (a) Use the formulas

$$s_1 = 1, \quad s_n = s_{n-1} + n \quad \text{for all } n \geq 2,$$

to write a recursive algorithm that computes

$$s_n = 1 + 2 + 3 + \cdots + n.$$

- (b) Give a proof using mathematical induction that your algorithm for part (a) is correct.
10. (a) Use the formulas

$$s_1 = 2, \quad s_n = s_{n-1} + 2n \quad \text{for all } n \geq 2,$$

to write a recursive algorithm that computes

$$s_n = 2 + 4 + 6 + \cdots + 2n.$$

- (b) Give a proof using mathematical induction that your algorithm for part (a) is correct.
11. (a) A robot can take steps of 1 meter, 2 meters, or 3 meters. Write a recursive algorithm to calculate the number of ways the robot can walk  $n$  meters.
- (b) Give a proof using mathematical induction that your algorithm for part (a) is correct.
12. Write a recursive algorithm to find the minimum of a finite sequence of numbers. Give a proof using mathematical induction that your algorithm is correct.
13. Write a recursive algorithm to find the maximum of a finite sequence of numbers. Give a proof using mathematical induction that your algorithm is correct.
14. Write a recursive algorithm that reverses a finite sequence. Give a proof using mathematical induction that your algorithm is correct.

15. Write a nonrecursive algorithm to compute  $n!$ .
- \*16. A robot can take steps of 1 meter or 2 meters. Write an algorithm to list all of the ways the robot can walk  $n$  meters.
- \*17. A robot can take steps of 1 meter, 2 meters, or 3 meters. Write an algorithm to list all of the ways the robot can walk  $n$  meters.

*Exercises 18–36 concern the Fibonacci sequence  $\{f_n\}$ .*

18. Suppose that at the beginning of the year, there is one pair of rabbits and that every month each pair produces a new pair that becomes productive after one month. Suppose further that no deaths occur. Let  $a_n$  denote the number of pairs of rabbits at the end of the  $n$ th month. Show that  $a_1 = 1$ ,  $a_2 = 2$ , and  $a_n - a_{n-1} = a_{n-2}$ . Prove that  $a_n = f_{n+1}$  for all  $n \geq 1$ .
19. Fibonacci's original question was: Under the conditions of Exercise 18, how many pairs of rabbits are there after one year? Answer Fibonacci's question.
20. Show that the number of ways to tile a  $2 \times n$  board with  $1 \times 2$  rectangular pieces is  $f_{n+1}$ , the  $(n+1)$ st Fibonacci number.

21. Use mathematical induction to show that

$$f_n^2 = f_{n-1}f_{n+1} + (-1)^{n+1} \quad \text{for all } n \geq 2.$$

22. Show that

$$f_n^2 = f_{n-2}f_{n+2} + (-1)^n \quad \text{for all } n \geq 3.$$

23. Show that

$$f_{n+2}^2 - f_{n+1}^2 = f_n f_{n+3} \quad \text{for all } n \geq 1.$$

24. Use mathematical induction to show that

$$\sum_{k=1}^n f_k^2 = f_n f_{n+1} \quad \text{for all } n \geq 1.$$

25. Use mathematical induction to show that for all  $n \geq 1$ ,  $f_n$  is even if and only if  $n$  is divisible by 3.
  - \*26. Use mathematical induction to show that
- $$f_{2n} = f_{n+1}^2 - f_{n-1}^2 \quad \text{and} \quad f_{2n+1} = f_n^2 + f_{n+1}^2 \quad \text{for all } n \geq 2.$$
27. Use mathematical induction to show that for all  $n \geq 6$ ,
- $$f_n > \left(\frac{3}{2}\right)^{n-1}.$$
28. Use mathematical induction to show that for all  $n \geq 1$ ,
- $$f_n \leq 2^{n-1}.$$

29. Use mathematical induction to show that for all  $n \geq 1$ ,

$$\sum_{k=1}^n f_{2k-1} = f_{2n}, \quad \sum_{k=1}^n f_{2k} = f_{2n+1} - 1.$$

- \*30. Use mathematical induction to show that every integer  $n \geq 1$  can be expressed as the sum of distinct Fibonacci numbers, no two of which are consecutive.
- \*31. Show that the representation in Exercise 30 is unique if we do not allow  $f_1$  as a summand.

32. Show that for all  $n \geq 2$ ,

$$f_n = \frac{f_{n-1} + \sqrt{5f_{n-1}^2 + 4(-1)^{n+1}}}{2}.$$

Notice that this formula gives  $f_n$  in terms of one predecessor rather than two predecessors as in the original definition.

33. Prove that

$$1 + \sum_{k=1}^n \frac{(-1)^{k+1}}{f_k f_{k+1}} = \frac{f_{n+2}}{f_{n+1}} \quad \text{for all } n \geq 1.$$

34. Define a sequence  $\{g_n\}$  as  $g_1 = c_1$  and  $g_2 = c_2$  for constants  $c_1$  and  $c_2$ , and

$$g_n = g_{n-1} + g_{n-2}$$

for  $n \geq 3$ . Prove that

$$g_n = g_1 f_{n-2} + g_2 f_{n-1}$$

for all  $n \geq 3$ .

35. Prove that

$$\sum_{k=1}^n (-1)^k f_k = (-1)^n f_{n-1} - 1 \quad \text{for all } n \geq 2.$$

36. Prove that

$$\sum_{k=1}^n (-1)^k k f_k = (-1)^n (n f_{n-1} + f_{n-3}) - 2 \quad \text{for all } n \geq 4.$$

37. [Requires calculus] Assume the formula for differentiating products:

$$\frac{d(fg)}{dx} = f \frac{dg}{dx} + g \frac{df}{dx}.$$

Use mathematical induction to prove that

$$\frac{dx^n}{dx} = nx^{n-1} \quad \text{for } n = 1, 2, \dots$$

38. [Requires calculus] Explain how the formula gives a recursive algorithm for integrating  $\log^n |x|$ :

$$\int \log^n |x| dx = x \log^n |x| - n \int \log^{n-1} |x| dx.$$

Give other examples of recursive integration formulas.

## Chapter 4 Notes

The first half of [Knuth, 1977] introduces the concept of an algorithm and various mathematical topics, including mathematical induction. The second half is devoted to data structures.

Most general references on computer science contain some discussion of algorithms. Books specifically on algorithms are [Aho; Baase; Brassard; Cormen; Johnsonbaugh; Knuth, 1997, 1998a, 1998b; Mamber; Miller; Nievergelt; and Reingold]. [McNaughton] contains a very thorough discussion on an introductory level of what an algorithm is. Knuth's expository article about algorithms ([Knuth, 1977]) and his article about the role of algorithms in the mathematical sciences ([Knuth, 1985]) are also recommended. [Gardner, 1992] contains a chapter about the Fibonacci sequence.

## Chapter 4 Review

### Section 4.1

1. Algorithm
2. Properties of an algorithm: Input, output, precision, determinism, finiteness, correctness, generality
3. Trace
4. Pseudocode

### Section 4.2

5. Searching
6. Text search
7. Text-search algorithm
8. Sorting

9. Insertion sort
10. Time and space for algorithms
11. Best-case time
12. Worst-case time
13. Randomized algorithm
14. Shuffle algorithm

### Section 4.3

15. Analysis of algorithms
16. Worst-case time of an algorithm
17. Best-case time of an algorithm
18. Average-case time of an algorithm

19. Big oh notation:  $f(n) = O(g(n))$
20. Omega notation:  $f(n) = \Omega(g(n))$
21. Theta notation:  $f(n) = \Theta(g(n))$

## Section 4.4

22. Recursive algorithm

23. Recursive function
24. Divide-and-conquer technique
25. Base cases: Situations where a recursive function does not invoke itself
26. Fibonacci sequence  $\{f_n\} : f_1 = 1, f_2 = 1, f_n = f_{n-1} + f_{n-2}, n \geq 3$

## Chapter 4 Self-Test

1. Trace Algorithm 4.1.1 for the values  $a = 12, b = 3, c = 0$ .
2. Which of the algorithm properties—input, output, precision, determinism, finiteness, correctness, generality—if any, are lacking in the following? Explain.

Input:  $S$  (a set of integers),  $m$  (an integer)

Output: All finite subsets of  $S$  that sum to  $m$

1. List all finite subsets of  $S$  and their sums.
2. Step through the subsets listed in 1 and output each whose sum is  $m$ .
3. Trace Algorithm 4.2.1 for the input  $t = "111011"$  and  $p = "110"$ .
4. Trace Algorithm 4.2.3 for the input 44, 64, 77, 15, 3.
5. Trace Algorithm 4.2.4 for the input 5, 51, 2, 44, 96.

Assume that the values of *rand* are

$$\begin{aligned} \text{rand}(1, 5) &= 1, & \text{rand}(2, 5) &= 3, & \text{rand}(3, 5) &= 5, \\ \text{rand}(4, 5) &= 5. \end{aligned}$$

6. Write an algorithm that receives as input the distinct numbers  $a, b$ , and  $c$  and assigns the values  $a, b$ , and  $c$  to the variables  $x, y$ , and  $z$  so that  $x < y < z$ .
7. Write an algorithm that receives as input the sequence  $s_1, \dots, s_n$  sorted in nondecreasing order and prints all values that appear more than once. *Example:* If the sequence is 1, 1, 1, 5, 8, 8, 9, 12, the output is 1 8.
8. Write an algorithm that returns true if the values of  $a, b$ , and  $c$  are distinct, and false otherwise.

9. Write an algorithm that tests whether two  $n \times n$  matrices are equal and find a theta notation for its worst-case time.

Select a theta notation from among  $\Theta(1), \Theta(n), \Theta(n^2), \Theta(n^3), \Theta(n^4), \Theta(2^n)$ , or  $\Theta(n!)$  for each of the expressions in Exercises 10 and 11.

10.  $4n^3 + 2n - 5$
11.  $1^3 + 2^3 + \dots + n^3$
12. Select a theta notation from among  $\Theta(1), \Theta(n), \Theta(n^2), \Theta(n^3), \Theta(2^n)$ , or  $\Theta(n!)$  for the number of times the line  $x = x + 1$  is executed.

```
for i = 1 to n
 for j = 1 to n
 x = x + 1
```

13. Trace Algorithm 4.4.4 (the tromino tiling algorithm) when  $n = 8$  and the missing square is four from the left and two from the top.

*Exercises 14–16 refer to the tribonacci sequence  $\{t_n\}$  defined by the equations*

$$t_1 = t_2 = t_3 = 1, \quad t_n = t_{n-1} + t_{n-2} + t_{n-3} \quad \text{for all } n \geq 4.$$

14. Find  $t_4$  and  $t_5$ .
15. Write a recursive algorithm to compute  $t_n, n \geq 1$ .
16. Give a proof using mathematical induction that your algorithm for Exercise 15 is correct.

## Chapter 4 Computer Exercises

1. Implement Algorithm 4.1.2, finding the largest element in a sequence, as a program.
2. Implement Algorithm 4.2.1, text search, as a program.
3. Implement Algorithm 4.2.3, insertion sort, as a program.
4. Implement Algorithm 4.2.4, shuffle, as a program.
5. Run shuffle (Algorithm 4.2.4) many times for the same input sequence. How might the output be analyzed to determine if it is truly “random”?
6. Implement selection sort (see Exercise 22, Section 4.2) as a program.
7. Compare the running times of insertion sort (Algorithm 4.2.3) and selection sort (see Exercise 22, Section 4.2) for several inputs of different sizes. Include data sorted in nondecreasing order, data sorted in nonincreasing order, data containing many duplicates, and data in random order.
8. Write recursive and nonrecursive programs to compute  $n!$ . Compare the times required by the programs.

9. Write a program whose input is a  $2^n \times 2^n$  board with one missing square and whose output is a tiling of the board by trominoes.
10. Write a program that uses a graphics display to show a tiling with trominoes of a  $2^n \times 2^n$  board with one square missing.
11. Write a program that tiles with trominoes an  $n \times n$  board with one square missing, provided that  $n \neq 5$  and 3 does not divide  $n$ .
12. Write recursive and nonrecursive programs to compute the Fibonacci sequence. Compare the times required by the programs.
13. A robot can take steps of 1 meter or 2 meters. Write a program to list all of the ways the robot can walk  $n$  meters.
14. A robot can take steps of 1, 2, or 3 meters. Write a program to list all of the ways the robot can walk  $n$  meters.



## Chapter 5

# INTRODUCTION TO NUMBER THEORY

- 
- 5.1** Divisors
  - 5.2** Representations of Integers and Integer Algorithms
  - 5.3** The Euclidean Algorithm
  - 5.4** The RSA Public-Key Cryptosystem

**Number theory** is the branch of mathematics concerned with the integers. Traditionally, number theory was a *pure* branch of mathematics—known for its abstract nature rather than its applications. The great English mathematician, G. H. Hardy (1877–1947), used number theory as an example of a beautiful, but impractical, branch of mathematics. However, in the late 1900s, number theory became extremely useful in cryptosystems—systems used for secure communications.

In the preceding chapters, we used some basic number theory definitions such as “divides” and “prime number.” In Section 5.1, we review these basic definitions and extend the discussion to unique factorization, greatest common divisors, and least common multiples.

In Section 5.2, we discuss representations of integers and some algorithms for integer arithmetic.

The Euclidean algorithm for computing the greatest common divisor is the subject of Section 5.3. This is surely one of the oldest algorithms. Euclid lived about 295 B.C., and the algorithm probably predates him.

As an application of the number theory presented in Sections 5.1–5.3, we discuss the RSA system for secure communications in Section 5.4.

---

## 5.1 Divisors

In this section, we give the basic definitions and terminology. We begin by recalling the definition of “divides,” and we introduce some related terminology.

**Definition 5.1.1** ► Let  $n$  and  $d$  be integers,  $d \neq 0$ . We say that  $d$  divides  $n$  if there exists an integer  $q$  satisfying  $n = dq$ . We call  $q$  the *quotient* and  $d$  a *divisor* or *factor* of  $n$ . If  $d$  divides  $n$ , we write  $d | n$ . If  $d$  does not divide  $n$ , we write  $d \nmid n$ .

---

### Example 5.1.2

Since  $21 = 3 \cdot 7$ , 3 divides 21 and we write  $3 | 21$ . The quotient is 7. We call 3 a divisor or factor of 21.

We note that if  $n$  and  $d$  are positive integers and  $d | n$ , then  $d \leq n$ . (If  $d \nmid n$ , there exists an integer  $q$  such that  $n = dq$ . Since  $n$  and  $d$  are positive integers,  $1 \leq q$ . Therefore,  $d \leq dq = n$ .)

Whether an integer  $d > 0$  divides an integer  $n$  or not, we obtain a unique quotient  $q$  and remainder  $r$  as given by the Quotient-Remainder Theorem (Theorem 2.5.6): There exist unique integers  $q$  (quotient) and  $r$  (remainder) satisfying  $n = dq + r$ ,  $0 \leq r < d$ . The remainder  $r$  equals zero if and only if  $d$  divides  $n$ .

Some additional properties of divisors are given in the following theorem and will be useful in our subsequent work in this chapter.

### Theorem 5.1.3

Let  $m$ ,  $n$ , and  $d$  be integers.

- (a) If  $d | m$  and  $d | n$ , then  $d | (m + n)$ .
- (b) If  $d | m$  and  $d | n$ , then  $d | (m - n)$ .
- (c) If  $d | m$ , then  $d | mn$ .

**Proof** (a) Suppose that  $d | m$  and  $d | n$ . By Definition 5.1.1,

$$m = dq_1 \quad (5.1.1)$$

for some integer  $q_1$  and

$$n = dq_2 \quad (5.1.2)$$

for some integer  $q_2$ . If we add equations (5.1.1) and (5.1.2), we obtain

$$m + n = dq_1 + dq_2 = d(q_1 + q_2).$$

Therefore,  $d$  divides  $m + n$  (with quotient  $q_1 + q_2$ ). We have proved part (a).

The proofs of parts (b) and (c) are left as exercises (see Exercises 27 and 28). ◀

**Definition 5.1.4 ►** An integer greater than 1 whose only positive divisors are itself and 1 is called *prime*. An integer greater than 1 that is not prime is called *composite*. ◀

### Example 5.1.5

The integer 23 is prime because its only divisors are itself and 1. The integer 34 is composite because it is divisible by 17, which is neither 1 nor 34. ◀

If an integer  $n > 1$  is composite, then it has a positive divisor  $d$  other than 1 and itself. Since  $d$  is positive and  $d \neq 1$ ,  $d > 1$ . Since  $d$  is a divisor of  $n$ ,  $d \leq n$ . Since  $d \neq n$ ,  $d < n$ . Therefore, to determine if a positive integer  $n$  is composite, it suffices to test whether any of the integers 2, 3, ...,  $n - 1$  divides  $n$ . If some integer in this list divides  $n$ , then  $n$  is composite. If no integer in this list divides  $n$ , then  $n$  is prime. (Actually, we can shorten this list considerably; see Theorem 5.1.7.)

### Example 5.1.6

By inspection, we find that none of 2, 3, 4, 5, ..., 41, 42 divides 43; thus, 43 is prime.

Checking the list 2, 3, 4, 5, ..., 449, 450 for potential divisors of 451, we find that 11 divides 451 ( $451 = 11 \cdot 41$ ); thus, 451 is composite. ◀

In Example 5.1.6, to determine whether a positive integer  $n > 1$  was prime, we checked the potential divisors 2, 3, ...,  $n - 1$ . Actually, it suffices to check only

$$2, 3, \dots, \lfloor \sqrt{n} \rfloor.$$

**Theorem 5.1.7**

A positive integer  $n$  greater than 1 is composite if and only if  $n$  has a divisor  $d$  satisfying  $2 \leq d \leq \sqrt{n}$ .

**Proof** We must prove

$$\text{If } n \text{ is composite, then } n \text{ has a divisor } d \text{ satisfying } 2 \leq d \leq \sqrt{n}, \quad (5.1.3)$$

and

$$\text{If } n \text{ has a divisor } d \text{ satisfying } 2 \leq d \leq \sqrt{n}, \text{ then } n \text{ is composite.} \quad (5.1.4)$$

We first prove (5.1.3). Suppose that  $n$  is composite. The discussion following Example 5.1.5 shows that  $n$  has a divisor  $d'$  satisfying  $2 \leq d' < n$ . We now argue by cases. If  $d' \leq \sqrt{n}$ , then  $n$  has a divisor  $d$  (namely  $d = d'$ ) satisfying  $2 \leq d \leq \sqrt{n}$ .

The other case is  $d' > \sqrt{n}$ . Since  $d'$  divides  $n$ , by Definition 5.1.1 there exists an integer  $q$  satisfying  $n = d'q$ . Thus  $q$  is also a divisor of  $n$ . We claim that  $q \leq \sqrt{n}$ . To show that  $q \leq \sqrt{n}$ , we use proof by contradiction. Thus, suppose that  $q > \sqrt{n}$ . Multiplying  $d' > \sqrt{n}$  and  $q > \sqrt{n}$  gives

$$n = d'q > \sqrt{n}\sqrt{n} = n,$$

which is a contradiction. Thus,  $q \leq \sqrt{n}$ . Therefore,  $n$  has a divisor  $d$  (namely  $d = q$ ) satisfying  $2 \leq d \leq \sqrt{n}$ .

It remains to prove (5.1.4). If  $n$  has a divisor  $d$  satisfying  $2 \leq d \leq \sqrt{n}$ , by Definition 5.1.4  $n$  is composite. The proof is complete. ◀

We may use Theorem 5.1.7 to construct the following algorithm that tests whether a positive integer  $n > 1$  is prime.

**Algorithm 5.1.8****Testing Whether an Integer Is Prime**

This algorithm determines whether the integer  $n > 1$  is prime. If  $n$  is prime, the algorithm returns 0. If  $n$  is composite, the algorithm returns a divisor  $d$  satisfying  $2 \leq d \leq \sqrt{n}$ . To test whether  $d$  divides  $n$ , the algorithm checks whether the remainder when  $n$  is divided by  $d$ ,  $n \bmod d$ , is zero.

```
Input: n
Output: d

is_prime(n) {
 for d = 2 to ⌊√n⌋
 if (n mod d == 0)
 return d
 return 0
}
```

**Go Online**

For a C++ program implementing this algorithm, see  
[goo.gl/y7T2se](http://goo.gl/y7T2se)

**Example 5.1.9**

To determine whether 43 is prime, Algorithm 5.1.8 checks whether any of 2, 3, 4, 5, 6 =  $\lfloor\sqrt{43}\rfloor$  divides 43. Since none of these numbers divides 43, the condition

$$n \bmod d == 0 \quad (5.1.5)$$

is always false. Therefore, the algorithm returns 0 to indicate that 43 is prime.

To determine whether 451 is prime, Algorithm 5.1.8 checks whether any of 2, 3, ..., 21 =  $\lfloor\sqrt{451}\rfloor$  divides 451. For  $d = 2, 3, \dots, 10$ ,  $d$  does not divide 451 and the condition (5.1.5) is false. However, when  $d = 11$ ,  $d$  does divide 451 and the condition

(5.1.5) is true. Therefore, the algorithm returns 11 to indicate that 451 is composite and 11 divides 451. 

In the worst case (when  $n$  is prime and the for loop runs to completion), Algorithm 5.1.8 takes time  $\Theta(\sqrt{n})$ . Although Algorithm 5.1.8 runs in time polynomial in  $n$  (since  $\sqrt{n} \leq n$ ), it does *not* run in time polynomial in the *size* of the input (namely,  $n$ ). [We can represent  $n$  in considerably less space than  $\Theta(n)$ ; see Example 5.2.1.] We say that Algorithm 5.1.8 is not a polynomial-time algorithm. It is not known whether there is a polynomial-time algorithm that can find a factor of a given integer; but most computer scientists think that there is no such algorithm. On the other hand, in 2002 Manindra Agrawal and two of his students, Nitin Saxena and Neeraj Kayal, discovered a polynomial-time algorithm that can determine whether or not a given integer is prime (see [Agarwal]). The question of whether there is a polynomial-time algorithm that can factor an integer is of more than academic interest since the security of certain encryption systems relies on the nonexistence of such an algorithm (see Section 5.4).

Notice that if a composite integer  $n$  is input to Algorithm 5.1.8, the divisor returned is prime; that is, Algorithm 5.1.8 returns a prime factor of a composite integer. To prove this, we use proof by contradiction. If Algorithm 5.1.8 returns a composite divisor of  $n$ , say  $a$ , then  $a$  has a divisor  $a'$  less than  $a$ . Since  $a'$  also divides  $n$  and  $a' < a$ , when Algorithm 5.1.8 sets  $d = a'$ , it will return  $a'$ , not  $a$ . This contradiction shows that if a composite integer  $n$  is input to Algorithm 5.1.8, the divisor returned is prime.

### Example 5.1.10

If the input to Algorithm 5.1.8 is  $n = 1274$ , the algorithm returns the prime 2 because 2 divides 1274, specifically  $1274 = 2 \cdot 637$ .

If we now input  $n = 637$  to Algorithm 5.1.8, the algorithm returns the prime 7 because 7 divides 637, specifically  $637 = 7 \cdot 91$ .

If we now input  $n = 91$  to Algorithm 5.1.8, the algorithm returns the prime 7 because 7 divides 91, specifically  $91 = 7 \cdot 13$ .

If we now input  $n = 13$  to Algorithm 5.1.8, the algorithm returns 0 because 13 is prime.

Combining the previous equations gives 1274 as a product of primes

$$1274 = 2 \cdot 637 = 2 \cdot 7 \cdot 91 = 2 \cdot 7 \cdot 7 \cdot 13.$$

We have illustrated how to write any integer greater than 1 as a product of primes. It is also a fact (although we will not prove it in this book) that, except for the order of the prime factors, the prime factors are unique. This result is known as the **Fundamental Theorem of Arithmetic** or the **unique factorization theorem**. (Unique factorization does *not* hold in some systems; see Exercise 44.) 

### Theorem 5.1.11

#### Fundamental Theorem of Arithmetic

Any integer greater than 1 can be written as a product of primes. Moreover, if the primes are written in nondecreasing order, the factorization is unique. In symbols, if

$$n = p_1 p_2 \cdots p_i,$$

where the  $p_k$  are primes and  $p_1 \leq p_2 \leq \cdots \leq p_i$ , and

$$n = p'_1 p'_2 \cdots p'_j,$$

where the  $p'_k$  are primes and  $p'_1 \leq p'_2 \leq \cdots \leq p'_j$ , then  $i = j$  and

$$p_k = p'_k \quad \text{for all } k = 1, \dots, i.$$

We next prove that the number of primes is infinite.

**Theorem 5.1.12**

The number of primes is infinite.

**Proof** It suffices to show that if  $p$  is a prime, there is a prime larger than  $p$ . To this end, we let  $p_1, p_2, \dots, p_n$  denote all of the distinct primes less than or equal to  $p$ . Consider the integer

$$m = p_1 p_2 \cdots p_n + 1.$$

Notice that when  $m$  is divided by  $p_i$ , the remainder is 1:

$$m = p_i q + 1, \quad q = p_1 p_2 \cdots p_{i-1} p_{i+1} \cdots p_n.$$

Therefore, for all  $i = 1$  to  $n$ ,  $p_i$  does *not* divide  $m$ . Let  $p'$  be a prime factor of  $m$  ( $m$  may or may not itself be prime; see Exercise 33). Then  $p'$  is not equal to any of  $p_i$ ,  $i = 1$  to  $n$ . Since  $p_1, p_2, \dots, p_n$  is a list of all of the primes less than or equal to  $p$ , we must have  $p' > p$ . The proof is complete. ▲

**Example 5.1.13**

Show how the proof of Theorem 5.1.12 produces a prime larger than 11.

**SOLUTION** We list the primes less than or equal to 11: 2, 3, 5, 7, 11. We let  $m = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 + 1 = 2311$ . Using Algorithm 5.1.8, we find that 2311 is prime. We have found a prime, namely 2311, larger than each of 2, 3, 5, 7, 11. (If 2311 had turned out not to be prime, Algorithm 5.1.8 would have found a factor of 2311, which would necessarily be larger than each of 2, 3, 5, 7, 11.) ▲

The **greatest common divisor** of two integers  $m$  and  $n$  (not both zero) is the largest positive integer that divides both  $m$  and  $n$ . For example, the greatest common divisor of 4 and 6 is 2, and the greatest common divisor of 3 and 8 is 1. We use the notion of greatest common divisor when we check to see if a fraction  $m/n$ , where  $m$  and  $n$  are integers, is in lowest terms. If the greatest common divisor of  $m$  and  $n$  is 1,  $m/n$  is in lowest terms; otherwise, we can reduce  $m/n$ . For example,  $4/6$  is not in lowest terms because the greatest common divisor of 4 and 6 is 2, not 1. (We can divide both 4 and 6 by 2.) The fraction  $3/8$  is in lowest terms because the greatest common divisor of 3 and 8 is 1.

**Definition 5.1.14** ► Let  $m$  and  $n$  be integers with not both  $m$  and  $n$  zero. A *common divisor* of  $m$  and  $n$  is an integer that divides both  $m$  and  $n$ . The *greatest common divisor*, written

$$\gcd(m, n),$$

is the largest common divisor of  $m$  and  $n$ . ▲

**Example 5.1.15**

The positive divisors of 30 are

$$1, 2, 3, 5, 6, 10, 15, 30,$$

and the positive divisors of 105 are

$$1, 3, 5, 7, 15, 21, 35, 105;$$

thus the positive common divisors of 30 and 105 are

$$1, 3, 5, 15.$$

It follows that the greatest common divisor of 30 and 105,  $\gcd(30, 105)$ , is 15. ▲

We can also find the greatest common divisor of two integers  $m$  and  $n$  by looking carefully at their prime factorizations. We illustrate with an example and then explain the technique in detail.

**Example 5.1.16**

We find the greatest common divisor of 30 and 105 by looking at their prime factorizations

$$30 = 2 \cdot 3 \cdot 5 \quad 105 = 3 \cdot 5 \cdot 7.$$

Notice that 3 is a common divisor of 30 and 105 since it occurs in the prime factorization of both numbers. For the same reason, 5 is also a common divisor of 30 and 105. Also,  $3 \cdot 5 = 15$  is also a common divisor of 30 and 105. Since no larger product of primes is common to both 30 and 105, we conclude that 15 is the greatest common divisor of 30 and 105.  $\blacktriangleleft$

We state the method of Example 5.1.16 as Theorem 5.1.17.

**Theorem 5.1.17**

Let  $m$  and  $n$  be integers,  $m > 1, n > 1$ , with prime factorizations

$$m = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$$

and

$$n = p_1^{b_1} p_2^{b_2} \cdots p_k^{b_k}.$$

(If the prime  $p_i$  is not a factor of  $m$ , we let  $a_i = 0$ . Similarly, if the prime  $p_i$  is not a factor of  $n$ , we let  $b_i = 0$ .) Then

$$\gcd(m, n) = p_1^{\min(a_1, b_1)} p_2^{\min(a_2, b_2)} \cdots p_k^{\min(a_k, b_k)}.$$

**Proof** Let  $g = \gcd(m, n)$ . We note that if a prime  $p$  appears in the prime factorization of  $g$ ,  $p$  must be equal to one of  $p_1, \dots, p_k$ ; otherwise,  $g$  would not divide  $m$  or  $n$  (or both). Therefore  $g = p_1^{c_1} \cdots p_k^{c_k}$  for some  $c_1, \dots, c_k$ . Now

$$p_1^{\min(a_1, b_1)} p_2^{\min(a_2, b_2)} \cdots p_k^{\min(a_k, b_k)} \tag{5.1.6}$$

divides both  $m$  and  $n$  and if any exponent,  $\min(a_i, b_i)$ , is increased, the resulting integer will fail to divide  $m$  or  $n$  (or both). Therefore (5.1.6) is the greatest common divisor of  $m$  and  $n$ .  $\blacktriangleleft$

**Example 5.1.18**

Using the notation of Theorem 5.1.17, we have

$$82320 = 2^4 \cdot 3^1 \cdot 5^1 \cdot 7^3 \cdot 11^0$$

and

$$950796 = 2^2 \cdot 3^2 \cdot 5^0 \cdot 7^4 \cdot 11^1.$$

By Theorem 5.1.17,

$$\begin{aligned} \gcd(82320, 950796) &= 2^{\min(4,2)} \cdot 3^{\min(1,2)} \cdot 5^{\min(1,0)} \cdot 7^{\min(3,4)} \cdot 11^{\min(0,1)} \\ &= 2^2 \cdot 3^1 \cdot 5^0 \cdot 7^3 \cdot 11^0 \\ &= 4116. \end{aligned} \quad \blacktriangleleft$$

Neither the “list all divisors” method of Example 5.1.15 nor use of prime factorization as in Example 5.1.18 is an efficient method of finding the greatest common divisor. The problem is that both methods require finding the prime factors of the numbers involved and no efficient algorithm is known to compute these prime factors. However, in Section 5.3, we will present the Euclidean algorithm, which does provide an efficient way to compute the greatest common divisor.

A companion to the greatest common divisor is the least common multiple.

**Definition 5.1.19 ▶** Let  $m$  and  $n$  be positive integers. A *common multiple* of  $m$  and  $n$  is an integer that is divisible by both  $m$  and  $n$ . The *least common multiple*, written

$$\text{lcm}(m, n),$$

is the smallest positive common multiple of  $m$  and  $n$ .

### Example 5.1.20

The least common multiple of 30 and 105,  $\text{lcm}(30, 105)$ , is 210 because 210 is divisible by both 30 and 105 and, by inspection, no positive integer smaller than 210 is divisible by both 30 and 105.

### Example 5.1.21

We can find the least common multiple of 30 and 105 by looking at their prime factorizations

$$30 = 2 \cdot 3 \cdot 5 \quad 105 = 3 \cdot 5 \cdot 7.$$

The prime factorization of  $\text{lcm}(30, 105)$  must contain 2, 3, and 5 as factors [so that 30 divides  $\text{lcm}(30, 105)$ ]. It must also contain 3, 5, and 7 [so that 105 divides  $\text{lcm}(30, 105)$ ]. The smallest number with this property is

$$2 \cdot 3 \cdot 5 \cdot 7 = 210.$$

Therefore,  $\text{lcm}(30, 105) = 210$ .

We state the method of Example 5.1.21 as Theorem 5.1.22.

### Theorem 5.1.22

Let  $m$  and  $n$  be integers,  $m > 1, n > 1$ , with prime factorizations

$$m = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$$

and

$$n = p_1^{b_1} p_2^{b_2} \cdots p_k^{b_k}.$$

(If the prime  $p_i$  is not a factor of  $m$ , we let  $a_i = 0$ . Similarly, if the prime  $p_i$  is not a factor of  $n$ , we let  $b_i = 0$ .) Then

$$\text{lcm}(m, n) = p_1^{\max(a_1, b_1)} p_2^{\max(a_2, b_2)} \cdots p_k^{\max(a_k, b_k)}.$$

**Proof** Let  $l = \text{lcm}(m, n)$ . We note that if a prime  $p$  appears in the prime factorization of  $l$ ,  $p$  must be equal to one of  $p_1, \dots, p_k$ ; otherwise, we could eliminate  $p$  and obtain a smaller integer that is divisible by both  $m$  and  $n$ . Therefore  $l = p_1^{c_1} \cdots p_k^{c_k}$  for some  $c_1, \dots, c_k$ . Now

$$p_1^{\max(a_1, b_1)} p_2^{\max(a_2, b_2)} \cdots p_k^{\max(a_k, b_k)} \quad (5.1.7)$$

is divisible by both  $m$  and  $n$  and if any exponent,  $\max(a_i, b_i)$ , is decreased, the resulting integer will fail to be divisible by  $m$  or  $n$  (or both). Therefore (5.1.7) is the least common multiple of  $m$  and  $n$ .  $\blacktriangleleft$

**Example 5.1.23** Using the notation of Theorem 5.1.22, we have

$$82320 = 2^4 \cdot 3^1 \cdot 5^1 \cdot 7^3 \cdot 11^0$$

and

$$950796 = 2^2 \cdot 3^2 \cdot 5^0 \cdot 7^4 \cdot 11^1.$$

By Theorem 5.1.22,

$$\begin{aligned} \text{lcm}(82320, 950796) &= 2^{\max(4,2)} \cdot 3^{\max(1,2)} \cdot 5^{\max(1,0)} \cdot 7^{\max(3,4)} \cdot 11^{\max(0,1)} \\ &= 2^4 \cdot 3^2 \cdot 5^1 \cdot 7^4 \cdot 11^1 \\ &= 19015920. \end{aligned}$$

**Example 5.1.24**

In Example 5.1.15, we found that  $\gcd(30, 105) = 15$ , and in Example 5.1.21, we found that  $\text{lcm}(30, 105) = 210$ . Notice that the product of the gcd and lcm is equal to the product of the pair of numbers; that is,

$$\gcd(30, 105) \cdot \text{lcm}(30, 105) = 15 \cdot 210 = 3150 = 30 \cdot 105.$$

This formula holds for any pair of numbers as we will show in Theorem 5.1.25.  $\blacktriangleleft$

**Theorem 5.1.25**

For any positive integers  $m$  and  $n$ ,

$$\gcd(m, n) \cdot \text{lcm}(m, n) = mn.$$

**Proof** If  $m = 1$ , then  $\gcd(m, n) = 1$  and  $\text{lcm}(m, n) = n$ , so

$$\gcd(m, n) \cdot \text{lcm}(m, n) = 1 \cdot n = mn.$$

Similarly, if  $n = 1$ , then  $\gcd(m, n) = 1$  and  $\text{lcm}(m, n) = m$ , so

$$\gcd(m, n) \cdot \text{lcm}(m, n) = 1 \cdot m = mn.$$

Thus, we may assume that  $m > 1$  and  $n > 1$ .

The proof combines the formulas for the gcd (Theorem 5.1.17) and lcm (Theorem 5.1.22) (which require that  $m > 1$  and  $n > 1$ ) with the fact that

$$\min(x, y) + \max(x, y) = x + y \quad \text{for all } x \text{ and } y.$$

This latter formula is true because one of  $\{\min(x, y), \max(x, y)\}$  equals  $x$  and the other equals  $y$ . We now put this all together to produce a proof.

Write the prime factorizations of  $m$  and  $n$  as

$$m = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$$

and

$$n = p_1^{b_1} p_2^{b_2} \cdots p_k^{b_k}.$$

(If the prime  $p_i$  is not a factor of  $m$ , we let  $a_i = 0$ . Similarly, if the prime  $p_i$  is not a factor of  $n$ , we let  $b_i = 0$ .) By Theorem 5.1.17,

$$\gcd(m, n) = p_1^{\min(a_1, b_1)} \cdots p_k^{\min(a_k, b_k)},$$

and by Theorem 5.1.22,

$$\text{lcm}(m, n) = p_1^{\max(a_1, b_1)} \cdots p_k^{\max(a_k, b_k)}.$$

Therefore,

$$\begin{aligned} \gcd(m, n) \cdot \text{lcm}(m, n) &= [p_1^{\min(a_1, b_1)} \cdots p_k^{\min(a_k, b_k)}] \cdot \\ &\quad [p_1^{\max(a_1, b_1)} \cdots p_k^{\max(a_k, b_k)}] \\ &= p_1^{\min(a_1, b_1) + \max(a_1, b_1)} \cdots p_k^{\min(a_k, b_k) + \max(a_k, b_k)} \\ &= p_1^{a_1+b_1} \cdots p_k^{a_k+b_k} \\ &= [p_1^{a_1} \cdots p_k^{a_k}] [p_1^{b_1} \cdots p_k^{b_k}] = mn. \end{aligned}$$

◀

If we have an algorithm to compute the greatest common divisor, we can compute the least common multiple by using Theorem 5.1.25:

$$\text{lcm}(m, n) = \frac{mn}{\gcd(m, n)}.$$

In particular, if we have an *efficient* algorithm to compute the greatest common divisor, we can efficiently compute the least common multiple as well.

### 5.1 Problem-Solving Tips

The straightforward way to determine whether an integer  $n > 1$  is prime is to test whether any of  $2, 3, \dots, \lfloor \sqrt{n} \rfloor$  divides  $n$ . While this technique becomes too time-consuming as  $n$  grows larger, it suffices for relatively small values of  $n$ . This technique can be iterated to find the prime factorization of  $n$ , again for relatively small values of  $n$ .

Two ways of finding the greatest common divisor of  $a$  and  $b$  were presented. The first way was to list all of the positive divisors of  $a$  and all of the positive divisors of  $b$  and then, among all of the common divisors, choose the largest. This technique is time consuming and was shown mainly to illustrate exactly what is meant by common divisors and the greatest common divisor.

The second technique was to compare the prime factorizations of  $a$  and  $b$ . If  $p^i$  appears in  $a$  and  $p^j$  appears in  $b$ , include  $p^{\min(i,j)}$  in the prime factorization of the greatest common divisor. This technique works well if the numbers  $a$  and  $b$  are relatively small so that the prime factorizations of each can be found, or if the prime factorizations of each are given. In Section 5.3, we present the Euclidean algorithm that efficiently finds the greatest common divisor even for large values of  $a$  and  $b$ .

If you compute the  $\gcd(a, b)$ , you can immediately compute the least common multiple using the formula

$$\text{lcm}(a, b) = \frac{ab}{\gcd(a, b)}.$$

The least common multiple can also be computed by comparing the prime factorizations of  $a$  and  $b$ . If  $p^i$  appears in  $a$  and  $p^j$  appears in  $b$ , include  $p^{\max(i,j)}$  in the prime factorization of the least common multiple.

## 5.1 Review Exercises

1. Define  $d$  divides  $n$ .
2. Define  $d$  is a divisor of  $n$ .
3. Define quotient.
4. Define  $n$  is prime.
5. Define  $n$  is composite.
6. Explain why, when testing whether an integer  $n > 1$  is prime by looking for divisors, we need only check whether any of 2 to  $\lfloor \sqrt{n} \rfloor$  divides  $n$ .
7. Explain why Algorithm 5.1.8 is not considered to be a polynomial-time algorithm.
8. What is the Fundamental Theorem of Arithmetic?
9. Prove that the number of primes is infinite.
10. What is a common divisor?
11. What is the greatest common divisor?
12. Explain how to compute the greatest common divisor of  $m$  and  $n$ , not both zero, given their prime factorizations.
13. What is a common multiple?
14. What is the least common multiple?
15. Explain how to compute the least common multiple of positive integers  $m$  and  $n$ , given their prime factorizations.
16. How are the greatest common divisor and least common multiple related?

## 5.1 Exercises

In Exercises 1–8, trace Algorithm 5.1.8 for the given input.

- |               |                  |
|---------------|------------------|
| 1. $n = 9$    | 2. $n = 209$     |
| 3. $n = 47$   | 4. $n = 637$     |
| 5. $n = 4141$ | 6. $n = 1007$    |
| 7. $n = 3738$ | 8. $n = 1050703$ |

9. Which of the integers in Exercises 1–8 are prime?
10. Find the prime factorization of each integer in Exercises 1–8.
11. Find the prime factorization of  $11!$ .

Find the greatest common divisor of each pair of integers in Exercises 12–24.

- |                                                      |                |               |
|------------------------------------------------------|----------------|---------------|
| 12. 0, 17                                            | 13. 60, 90     | 14. 5, 25     |
| 15. 110, 273                                         | 16. 315, 825   | 17. 220, 1400 |
| 18. 20, 40                                           | 19. 2091, 4807 | 20. 331, 993  |
| 21. 13, $13^2$                                       | 22. 15, $15^9$ |               |
| 23. $3^2 \cdot 7^3 \cdot 11, 2^3 \cdot 5 \cdot 7$    |                |               |
| 24. $3^2 \cdot 7^3 \cdot 11, 3^2 \cdot 7^3 \cdot 11$ |                |               |

25. Find the least common multiple of each pair of integers in Exercises 13–24.
26. For each pair of integers in Exercises 13–24, verify that  $\gcd(m, n) \cdot \text{lcm}(m, n) = mn$ .
27. Let  $m$ ,  $n$ , and  $d$  be integers. Show that if  $d \mid m$  and  $d \mid n$ , then  $d \mid (m - n)$ .
28. Let  $m$ ,  $n$ , and  $d$  be integers. Show that if  $d \mid m$ , then  $d \mid mn$ .
29. Let  $m$ ,  $n$ ,  $d_1$ , and  $d_2$  be integers. Show that if  $d_1 \mid m$  and  $d_2 \mid n$ , then  $d_1 d_2 \mid mn$ .
30. Let  $n$ ,  $c$ , and  $d$  be integers. Show that if  $dc \mid nc$ , then  $d \mid n$ .

31. Let  $a$ ,  $b$ , and  $c$  be integers. Show that if  $a \mid b$  and  $b \mid c$ , then  $a \mid c$ .

32. Suggest ways to make Algorithm 5.1.8 more efficient.
33. Give an example of consecutive primes  $p_1 = 2, p_2, \dots, p_n$  where

$$p_1 p_2 \cdots p_n + 1$$

is not prime.

Exercises 34 and 35 use the following definition: A subset  $\{a_1, \dots, a_n\}$  of  $\mathbb{Z}^+$  is a \*-set of size  $n$  if  $(a_i - a_j) \mid a_i$  for all  $i$  and  $j$ , where  $i \neq j$ ,  $1 \leq i \leq n$ , and  $1 \leq j \leq n$ . These exercises are due to Martin Gilchrist.

34. Prove that for all  $n \geq 2$ , there exists a \*-set of size  $n$ . Hint: Use induction on  $n$ . For the Basis Step, consider the set  $\{1, 2\}$ . For the Inductive Step, let  $b_0 = \prod_{k=1}^n a_k$  and  $b_i = b_0 + a_i$  for  $1 \leq i \leq n$ .
35. Using the hint in Exercise 34, construct \*-sets of sizes 3 and 4.

The Fermat numbers  $F_0, F_1, \dots$  are defined as  $F_n = 2^{2^n} + 1$ .

36. Prove that

$$\prod_{i=0}^{n-1} F_i = F_n - 2 \quad \text{for all } n, n \geq 1.$$

37. Using Exercise 36 or otherwise, prove that

$$\gcd(F_m, F_n) = 1 \quad \text{for all } m, n, 0 \leq m < n.$$

38. Use Exercise 37 to prove that the number of primes is infinite.
39. Recall that a Mersenne prime (see the discussion before Example 2.2.14) is a prime of the form  $2^p - 1$ , where  $p$  is prime. Prove that if  $m$  is composite,  $2^m - 1$  is also composite.

Exercises 40–49 use the following notation and terminology. We let  $E$  denote the set of positive, even integers. If  $n \in E$  can be written as a product of two or more elements in  $E$ , we say that  $n$  is  $E$ -composite; otherwise, we say that  $n$  is  $E$ -prime. As examples, 4 is  $E$ -composite and 6 is  $E$ -prime.

40. Is 2  $E$ -prime or  $E$ -composite?
41. Is 8  $E$ -prime or  $E$ -composite?
42. Is 10  $E$ -prime or  $E$ -composite?
43. Is 12  $E$ -prime or  $E$ -composite?
44. Show that the number 36 can be written as a product of  $E$ -primes in two different ways, which shows that factoring into  $E$ -primes is *not* necessarily unique.

45. Find a necessary and sufficient condition for an integer to be an  $E$ -prime. Prove your statement.
46. Show that the set of  $E$ -primes is infinite.
47. Show that there are no twin  $E$ -primes, that is, two  $E$ -primes that differ by 2.
48. Show that there are infinitely many pairs of  $E$ -primes that differ by 4.
49. Give an example to show that the following is false: If an  $E$ -prime  $p$  divides  $mn \in E$ , then  $p$  divides  $m$  or  $p$  divides  $n$ . “Divides” means “divides in  $E$ .” That is, if  $p, q \in E$ , we say that  $p$  divides  $q$  in  $E$  if  $q = pr$ , where  $r \in E$ . (Compare this result with Exercise 27, Section 5.3.)

## 5.2

# Representations of Integers and Integer Algorithms

### Go Online

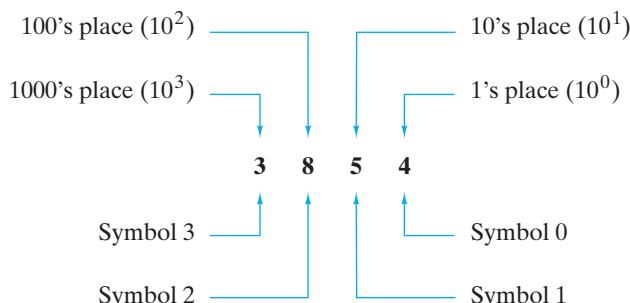
For more on representations of integers, see  
[goo.gl/WyqJp9](http://goo.gl/WyqJp9)

A **bit** is a *binary digit*, that is, a 0 or a 1. In a digital computer, data and instructions are encoded as bits. (The term *digital* refers to the use of the digits 0 and 1.) Technology determines how the bits are physically represented within a computer system. Today’s hardware relies on the state of an electronic circuit to represent a bit. The circuit must be capable of being in two states—one representing 1, the other 0. In this section we discuss the **binary number system**, which represents integers using bits, and the **hexadecimal number system**, which represents integers using 16 symbols. The **octal number system**, which represents integers using eight symbols, is discussed before Exercise 42.

In the decimal number system, to represent integers we use the 10 symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. In representing an integer, the symbol’s position is significant; reading from the right, the first symbol represents the number of 1’s, the next symbol the number of 10’s, the next symbol the number of 100’s, and so on. For example,

$$3854 = 3 \cdot 10^3 + 8 \cdot 10^2 + 5 \cdot 10^1 + 4 \cdot 10^0$$

(see Figure 5.2.1). In general, the symbol in position  $n$  (with the rightmost symbol being in position 0) represents the number of  $10^n$ ’s. Since  $10^0 = 1$ , the symbol in position 0 represents the number of  $10^0$ ’s or 1’s; since  $10^1 = 10$ , the symbol in position 1 represents the number of  $10^1$ ’s or 10’s; since  $10^2 = 100$ , the symbol in position 2 represents the number of  $10^2$ ’s or 100’s; and so on. We call the value on which the system is based (10 in the case of the decimal system) the **base** of the number system.

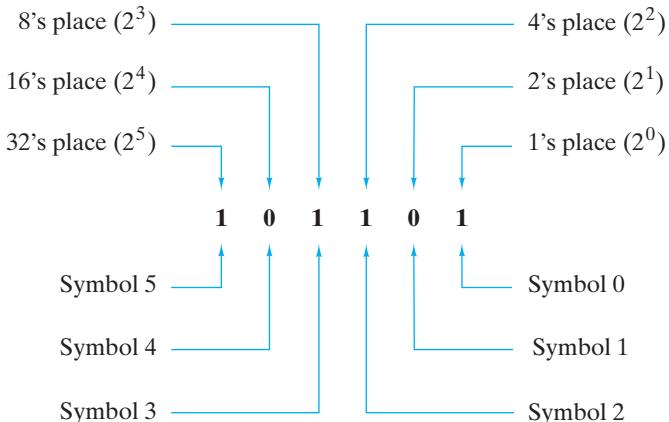


**Figure 5.2.1** The decimal number system.

In the binary (base 2) number system, to represent integers we need only two symbols, 0 and 1. In representing an integer, reading from the right, the first symbol represents the number of 1's, the next symbol the number of 2's, the next symbol the number of 4's, the next symbol the number of 8's, and so on. For example, in base 2,<sup>†</sup>

$$101101_2 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

(see Figure 5.2.2). In general, the symbol in position  $n$  (with the rightmost symbol being in position 0) represents the number of  $2^n$ 's. Since  $2^0 = 1$ , the symbol in position 0 represents the number of  $2^0$ 's, or 1's; since  $2^1 = 2$ , the symbol in position 1 represents the number of  $2^1$ 's or 2's; since  $2^2 = 4$ , the symbol in position 2 represents the number of  $2^2$ 's or 4's; and so on.



**Figure 5.2.2** The binary number system.

### Example 5.2.1

**Computer Representation of Integers** Computer systems represent integers in binary. Compute the number of bits necessary to represent a positive integer  $n$ . Deduce that Algorithm 5.1.8, which determines whether an integer is prime, is not a polynomial-time algorithm.

**SOLUTION** Suppose that the binary representation of the positive  $k$ -bit integer  $n$  is

$$n = 1 \cdot 2^{k-1} + b_{k-2}2^{k-2} + \cdots + b_02^0.$$

Now

$$2^{k-1} \leq n$$

and

$$\begin{aligned} n &= 1 \cdot 2^{k-1} + b_{k-2}2^{k-2} + \cdots + b_02^0 \\ &\leq 1 \cdot 2^{k-1} + 1 \cdot 2^{k-2} + \cdots + 1 \cdot 2^0 = 2^k - 1 < 2^k. \end{aligned}$$

(The last equality follows from the formula for the geometric sum; see Example 2.4.4.) Therefore

$$2^{k-1} \leq n < 2^k.$$

---

<sup>†</sup>Without knowing which number system is being used, a representation is ambiguous; for example, 101101 represents one number in decimal and quite a different number in binary. Often the context will make clear which number system is in effect; but when we want to be absolutely clear, we subscript the number to specify the base—the subscript 10 denotes the decimal system and the subscript 2 denotes the binary system.

Taking logs, we obtain

$$k - 1 \leq \lg n < k.$$

Adding one gives

$$k \leq 1 + \lg n < k + 1.$$

Therefore,  $k = \lfloor 1 + \lg n \rfloor$ . Since  $k$  is the number of bits required to represent  $n$ , we have proved that the number of bits necessary to represent  $n$  is  $\lfloor 1 + \lg n \rfloor$ .

The size  $s$  of an integer  $n$  input to Algorithm 5.1.8 is the number of bits necessary to represent  $n$ . Thus  $s$  satisfies

$$s = \lfloor 1 + \lg n \rfloor \leq 1 + \lg n = \lg 2 + \lg n = \lg(2n).$$

Raising to the power 2 gives  $2^s \leq 2n$ . Dividing by 2 and taking square roots yields

$$\frac{1}{\sqrt{2}}(\sqrt{2})^s \leq \sqrt{n}. \quad (5.2.1)$$

The worst-case time of Algorithm 5.1.8 is  $\Theta(\sqrt{n})$ . Thus its worst-case time  $T$  satisfies

$$T \geq C\sqrt{n} \quad (5.2.2)$$

for some constant  $C$ . Combining inequalities (5.2.1) and (5.2.2), we obtain  $T \geq (C/\sqrt{2})(\sqrt{2})^s$ . Therefore, in the worst case, Algorithm 5.1.8 runs in *exponential time* in the input size  $s$ . We say that Algorithm 5.1.8 is *not* a polynomial-time algorithm. ◀

### Example 5.2.2

**Binary to Decimal** The binary number  $101101_2$  represents the number consisting of one 1, no 2's, one 4, one 8, no 16's, and one 32 (see Figure 5.2.2). This representation may be expressed

$$101101_2 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0.$$

Computing the right-hand side in decimal, we find that

$$\begin{aligned} 101101_2 &= 1 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 \\ &= 32 + 8 + 4 + 1 = 45_{10}. \end{aligned}$$



We turn the method of Example 5.2.2 into an algorithm. We generalize by allowing an arbitrary base  $b$ .

### Algorithm 5.2.3

#### Converting an Integer from Base $b$ to Decimal

This algorithm returns the decimal value of the base  $b$  integer  $c_n c_{n-1} \cdots c_1 c_0$ .

```

Input: c, n, b
Output: dec_val

base_b_to_dec (c, n, b) {
 dec_val = 0
 b_to_the_i = 1
 for i = 0 to n {
 dec_val = dec_val + c_i * b_to_the_i
 b_to_the_i = b_to_the_i * b
 }
 return dec_val
}

```

Algorithm 5.2.3 runs in time  $\Theta(n)$ .

### Example 5.2.4

Show how Algorithm 5.2.3 converts the binary number 1101 to decimal.

**SOLUTION** Here  $n = 3$ ,  $b = 2$ , and

$$c_3 = 1, \quad c_2 = 1, \quad c_1 = 0, \quad c_0 = 1.$$

First,  $dec\_val$  is set to 0, and  $b\_to\_the\_i$  is set to 1. We then enter the for loop.

Since  $i = 0$  and  $b\_to\_the\_i = 1$ ,

$$c_i * b\_to\_the\_i = 1 * 1 = 1.$$

Thus  $dec\_val$  becomes 1. Executing

$$b\_to\_the\_i = b\_to\_the\_i * b$$

sets  $b\_to\_the\_i$  to 2. We return to the top of the for loop.

Since  $i = 1$  and  $b\_to\_the\_i = 2$ ,

$$c_i * b\_to\_the\_i = 0 * 2 = 0.$$

Thus  $dec\_val$  remains 1. Executing

$$b\_to\_the\_i = b\_to\_the\_i * b$$

sets  $b\_to\_the\_i$  to 4. We return to the top of the for loop.

Since  $i = 2$  and  $b\_to\_the\_i = 4$ ,

$$c_i * b\_to\_the\_i = 1 * 4 = 4.$$

Thus  $dec\_val$  becomes 5. Executing

$$b\_to\_the\_i = b\_to\_the\_i * b$$

sets  $b\_to\_the\_i$  to 8. We return to the top of the for loop.

Since  $i = 3$  and  $b\_to\_the\_i = 8$ ,

$$c_i * b\_to\_the\_i = 1 * 8 = 8.$$

Thus  $dec\_val$  becomes 13. Executing

$$b\_to\_the\_i = b\_to\_the\_i * b$$

sets  $b\_to\_the\_i$  to 16. The for loop terminates and the algorithm returns 13, the decimal value of the binary number 1101. 

Other important bases for number systems in computer science are base 8 or **octal** and base 16 or **hexadecimal** (sometimes shortened to **hex**). We will discuss the hexadecimal system and leave the octal system to the exercises (see Exercises 45–50).

In the hexadecimal number system, to represent integers we use the symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. The symbols A–F are interpreted as decimal 10–15. (In general, in the base  $N$  number system,  $N$  distinct symbols, representing 0, 1, 2, ...,  $N - 1$  are required.) In representing an integer, reading from the right, the first symbol represents the number of 1's, the next symbol the number of 16's, the next symbol the number of 16<sup>2</sup>'s, and so on. For example, in base 16,

$$B4F = 11 \cdot 16^2 + 4 \cdot 16^1 + 15 \cdot 16^0$$

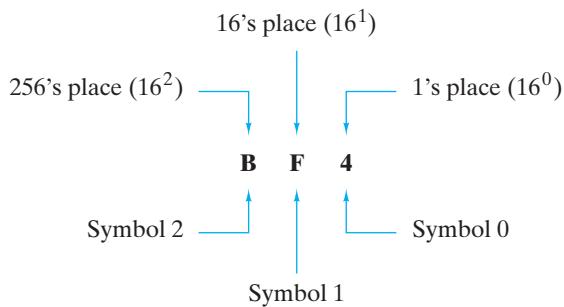


Figure 5.2.3 The hexadecimal number system.

(see Figure 5.2.3). In general, the symbol in position  $n$  (with the rightmost symbol being in position 0) represents the number of  $16^n$ 's.

**Example 5.2.5**

**Hexadecimal to Decimal** Convert the hexadecimal number B4F to decimal.

**SOLUTION** We obtain

$$\begin{aligned} B4F_{16} &= 11 \cdot 16^2 + 4 \cdot 16^1 + 15 \cdot 16^0 \\ &= 11 \cdot 256 + 4 \cdot 16 + 15 = 2816 + 64 + 15 = 2895_{10}. \end{aligned}$$



Algorithm 5.2.3 shows how to convert an integer in base  $b$  to decimal. Consider the reverse problem—converting a decimal number to base  $b$ . Suppose, for example, that we want to convert the decimal number 91 to binary. If we divide 91 by 2, we obtain

$$\begin{array}{r} 45 \\ 2)91 \\ \underline{-8} \\ 11 \\ \underline{-10} \\ 1 \end{array}$$

This computation shows that

$$91 = 2 \cdot 45 + 1. \quad (5.2.3)$$

We are beginning to express 91 in powers of 2. If we next divide 45 by 2, we find

$$45 = 2 \cdot 22 + 1. \quad (5.2.4)$$

Substituting this expression for 45 into (5.2.3), we obtain

$$\begin{aligned} 91 &= 2 \cdot 45 + 1 \\ &= 2 \cdot (2 \cdot 22 + 1) + 1 \\ &= 2^2 \cdot 22 + 2 + 1. \end{aligned} \quad (5.2.5)$$

If we next divide 22 by 2, we find

$$22 = 2 \cdot 11.$$

Substituting this expression for 22 into (5.2.5), we obtain

$$\begin{aligned} 91 &= 2^2 \cdot 22 + 2 + 1 \\ &= 2^2 \cdot (2 \cdot 11) + 2 + 1 \\ &= 2^3 \cdot 11 + 2 + 1. \end{aligned} \quad (5.2.6)$$

If we next divide 11 by 2, we find

$$11 = 2 \cdot 5 + 1.$$

Substituting this expression for 11 into (5.2.6), we obtain

$$91 = 2^4 \cdot 5 + 2^3 + 2 + 1. \quad (5.2.7)$$

If we next divide 5 by 2, we find

$$5 = 2 \cdot 2 + 1.$$

Substituting this expression for 5 into (5.2.7), we obtain

$$\begin{aligned} 91 &= 2^5 \cdot 2 + 2^4 + 2^3 + 2 + 1 \\ &= 2^6 + 2^4 + 2^3 + 2 + 1 \\ &= 1011011_2. \end{aligned}$$

The preceding computation shows that the *remainders*, as  $N$  is successively divided by 2, give the bits in the binary representation of  $N$ . The first division by 2 in (5.2.3) gives the 1's bit; the second division by 2 in (5.2.4) gives the 2's bit; and so on. We illustrate with another example.

### Example 5.2.6

**Decimal to Binary** Write the decimal number 130 in binary.

**SOLUTION** The computation shows the successive divisions by 2 with the remainders recorded at the right.

|               |               |           |
|---------------|---------------|-----------|
| 2) <u>130</u> | remainder = 0 | 1's bit   |
| 2) <u>65</u>  | remainder = 1 | 2's bit   |
| 2) <u>32</u>  | remainder = 0 | 4's bit   |
| 2) <u>16</u>  | remainder = 0 | 8's bit   |
| 2) <u>8</u>   | remainder = 0 | 16's bit  |
| 2) <u>4</u>   | remainder = 0 | 32's bit  |
| 2) <u>2</u>   | remainder = 0 | 64's bit  |
| 2) <u>1</u>   | remainder = 1 | 128's bit |
| 0             |               |           |

We may stop when the quotient is 0. Remembering that the first remainder gives the number of 1's, the second remainder gives the number of 2's, and so on, we obtain

$$130_{10} = 10000010_2.$$

We turn the method of Example 5.2.6 into an algorithm. We generalize by allowing an arbitrary base  $b$ .

### Algorithm 5.2.7

#### Converting a Decimal Integer into Base $b$

This algorithm converts the positive integer  $m$  into the base  $b$  integer  $c_n c_{n-1} \cdots c_1 c_0$ . The variable  $n$  is used as an index in the sequence  $c$ . The value of  $m \bmod b$  is the remainder when  $m$  is divided by  $b$ . The value of  $\lfloor m/b \rfloor$  is the quotient when  $m$  is divided by  $b$ .

```

Input: m, b
Output: c, n

dec_to_base_b(m, b, c, n) {
 $n = -1$
 while ($m > 0$) {
 $n = n + 1$
 $c_n = m \bmod b$
 $m = \lfloor m/b \rfloor$
 }
}

```

Just as a binary integer  $m$  has  $\lfloor 1 + \lg m \rfloor$  bits, a base  $b$  integer  $m$  has  $\lfloor 1 + \log_b m \rfloor$  digits (see Exercise 55). Thus Algorithm 5.2.7 runs in time  $\Theta(\log_b m)$ .

### Example 5.2.8

Show how Algorithm 5.2.7 converts the decimal number  $m = 11$  to binary.

**SOLUTION** The algorithm first sets  $n$  to  $-1$ . The first time we arrive at the while loop,  $m = 11$  and the condition  $m > 0$  is true; so we execute the body of the while loop. The variable  $n$  is incremented and becomes  $0$ . Since  $m \bmod b = 11 \bmod 2 = 1$ ,  $c_0$  is set to  $1$ . Since  $\lfloor m/b \rfloor = \lfloor 11/2 \rfloor = 5$ ,  $m$  is set to  $5$ . We return to the top of the while loop.

Since  $m = 5$ , the condition  $m > 0$  is true; so we execute the body of the while loop. The variable  $n$  is incremented and becomes  $1$ . Since  $m \bmod b = 5 \bmod 2 = 1$ ,  $c_1$  is set to  $1$ . Since  $\lfloor m/b \rfloor = \lfloor 5/2 \rfloor = 2$ ,  $m$  is set to  $2$ . We return to the top of the while loop.

Since  $m = 2$ , the condition  $m > 0$  is true; so we execute the body of the while loop. The variable  $n$  is incremented and becomes  $2$ . Since  $m \bmod b = 2 \bmod 2 = 0$ ,  $c_2$  is set to  $0$ . Since  $\lfloor m/b \rfloor = \lfloor 2/2 \rfloor = 1$ ,  $m$  is set to  $1$ . We return to the top of the while loop.

Since  $m = 1$ , the condition  $m > 0$  is true; so we execute the body of the while loop. The variable  $n$  is incremented and becomes  $3$ . Since  $m \bmod b = 1 \bmod 2 = 1$ ,  $c_3$  is set to  $1$ . Since  $\lfloor m/b \rfloor = \lfloor 1/2 \rfloor = 0$ ,  $m$  is set to  $0$ . We return to the top of the while loop.

Since  $m = 0$ , the algorithm terminates. The value  $11$  has been converted to the binary number  $c_3c_2c_1c_0 = 1011$ . ◀

### Example 5.2.9

**Decimal to Hexadecimal** Convert the decimal number  $20385$  to hexadecimal.

**SOLUTION** The computation shows the successive divisions by  $16$  with the remainders recorded at the right.

$$\begin{array}{r}
 16) \underline{20385} \quad \text{remainder} = 1 \quad 1\text{'s place} \\
 16) \underline{1274} \quad \text{remainder} = 10 \quad 16\text{'s place} \\
 16) \underline{79} \quad \text{remainder} = 15 \quad 16^2\text{'s place} \\
 16) \underline{4} \quad \text{remainder} = 4 \quad 16^3\text{'s place} \\
 0
 \end{array}$$

We stop when the quotient is  $0$ . The first remainder gives the number of  $1$ 's, the second remainder gives the number of  $16$ 's, and so on; thus we obtain  $20385_{10} = 4FA1_{16}$ . ◀

Next we turn our attention to addition of numbers in arbitrary bases. The same method that we use to add decimal numbers can be used to add binary numbers; however, we must replace the decimal addition table with the binary addition table

|   |   |    |
|---|---|----|
|   | 0 | 1  |
| 0 | 0 | 1  |
| 1 | 1 | 10 |

(In decimal,  $1 + 1 = 2$ , and  $2_{10} = 10_2$ ; thus, in binary,  $1 + 1 = 10$ .)

---

**Example 5.2.10** **Binary Addition** Add the binary numbers  $10011011$  and  $1011011$ .

---

**SOLUTION** We write the problem as

$$\begin{array}{r} 10011011 \\ + \underline{1011011} \end{array}$$

As in decimal addition, we begin from the right, adding 1 and 1. This sum is  $10_2$ ; thus we write 0 and carry 1. At this point the computation is

$$\begin{array}{r} & 1 \\ & 10011011 \\ + & \underline{1011011} \\ & 0 \end{array}$$

Next, we add 1 and 1 and 1, which is  $11_2$ . We write 1 and carry 1. At this point, the computation is

$$\begin{array}{r} & 1 \\ & 10011011 \\ + & \underline{1011011} \\ & 10 \end{array}$$

Continuing in this way, we obtain

$$\begin{array}{r} 10011011 \\ + \underline{1011011} \\ 11110110 \end{array}$$

---

**Example 5.2.11** The addition problem of Example 5.2.10, in decimal, is

---

$$\begin{array}{r} 155 \\ + \underline{91} \\ 246 \end{array}$$

We turn the method of Example 5.2.10 into an algorithm. If the numbers to add are  $b_n b_{n-1} \cdots b_1 b_0$  and  $b'_n b'_{n-1} \cdots b'_1 b'_0$ , at the iteration  $i > 0$  the algorithm adds  $b_i$ ,  $b'_i$ , and the carry bit from the previous iteration. When adding three bits, say  $B_1$ ,  $B_2$ , and  $B_3$ , we obtain a two-bit binary number, say  $cb$ . For example, if we compute  $1 + 0 + 1$ , the result is  $10_2$ ; in our notation,  $c = 1$  and  $b = 0$ . By checking the various cases, we can verify that we can compute the binary sum  $B_1 + B_2 + B_3$  by first computing the sum in decimal and then recovering  $c$  and  $b$  from the formulas

$$b = (B_1 + B_2 + B_3) \bmod 2, \quad c = \lfloor (B_1 + B_2 + B_3)/2 \rfloor.$$

**Algorithm 5.2.12****Adding Binary Numbers**

This algorithm adds the binary numbers  $b_n b_{n-1} \dots b_1 b_0$  and  $b'_n b'_{n-1} \dots b'_1 b'_0$  and stores the sum in  $s_{n+1} s_n s_{n-1} \dots s_1 s_0$ . (Leading zeros can be appended to  $b$  or  $b'$  so that this algorithm can be used to add integers with different numbers of bits.)

Input:  $b, b', n$

Output:  $s$

```
binary-addition(b, b', n, s) {
 carry = 0
 for i = 0 to n {
 si = (bi + b'i + carry) mod 2
 carry = ⌊(bi + b'i + carry)/2⌋
 }
 sn+1 = carry
}
```

Algorithm 5.2.12 runs in time  $\Theta(n)$ .

Our next example shows that we can add hexadecimal numbers in the same way that we add decimal or binary numbers.

**Example 5.2.13**

**Hexadecimal Addition** Add the hexadecimal numbers 84F and 42EA.

**SOLUTION** The problem may be written

$$\begin{array}{r} 84F \\ + 42EA \\ \hline \end{array}$$

We begin in the rightmost column by adding F and A. Since F is  $15_{10}$  and A is  $10_{10}$ ,  $F + A = 15_{10} + 10_{10} = 25_{10} = 19_{16}$ . We write 9 and carry 1:

$$\begin{array}{r} & 1 \\ & 84F \\ + & 42EA \\ \hline & 9 \end{array}$$

Next, we add 1, 4, and E, obtaining  $13_{16}$ . We write 3 and carry 1:

$$\begin{array}{r} & 1 \\ & 84F \\ + & 42EA \\ \hline & 39 \end{array}$$

Continuing in this way, we obtain

$$\begin{array}{r} 84F \\ + 42EA \\ \hline 4B39 \end{array}$$

**Example 5.2.14**

The addition problem of Example 5.2.13, in decimal, is

$$\begin{array}{r} 2127 \\ + 17130 \\ \hline 19257 \end{array}$$

We can multiply binary numbers by modifying the standard algorithm for multiplying decimal numbers (see Exercise 67).

We conclude by discussing a special algorithm, which we will need in Section 5.4, to compute powers mod  $z$ . We first discuss an algorithm to compute a power  $a^n$  (without dealing with mod  $z$ ). The straightforward way to compute this power is to repeatedly multiply by  $a$

$$\underbrace{a \cdot a \cdots a}_{n \text{ } a's},$$

which uses  $n - 1$  multiplications. We can do better using **repeated squaring**.

As a concrete example, consider computing  $a^{29}$ . We first compute  $a^2 = a \cdot a$ , which uses 1 multiplication. We next compute  $a^4 = a^2 \cdot a^2$ , which uses 1 additional multiplication. We next compute  $a^8 = a^4 \cdot a^4$ , which uses 1 additional multiplication. We next compute  $a^{16} = a^8 \cdot a^8$ , which uses 1 additional multiplication. So far, we have used only 4 multiplications. Noting that the expansion of 29 in powers of 2, that is the binary expansion, is

$$29 = 1 + 4 + 8 + 16,$$

we see that we can compute  $a^{29}$  as

$$a^{29} = a^1 \cdot a^4 \cdot a^8 \cdot a^{16},$$

which uses 3 additional multiplications for a total of 7 multiplications. The straightforward technique uses 28 multiplications.

In Example 5.2.6, we saw that the remainders when  $n$  is successively divided by 2 give the binary expansion of  $n$ . If the remainder is 1, the corresponding power of 2 is included; otherwise, it is not included. We can formalize the repeated squaring technique if, in addition to repeated squaring, we simultaneously determine the binary expansion of the exponent.

### Example 5.2.15

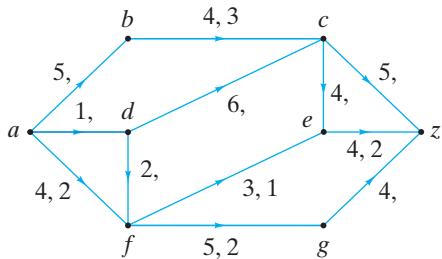
Figure 5.2.4 shows how  $a^{29}$  is calculated using repeated squaring. Initially  $x$  is set to  $a$ , and  $n$  is set to the value of the exponent, 29 in this case. We then compute  $n \bmod 2$ . Since this value is 1, we know that  $1 = 2^0$  is included in the binary expansion of 29. Therefore  $a^1$  is included in the product. We track the partial product in *Result*; so *Result* is set to  $a$ . We then compute the quotient when 29 is divided by 2. The quotient 14 becomes the new value of  $n$ . We then repeat this process.

| <i>x</i> | <i>Current Value<br/>of n</i> | <i>n mod 2</i> | <i>Result</i>                  | <i>Quotient When n<br/>Divided by 2</i> |
|----------|-------------------------------|----------------|--------------------------------|-----------------------------------------|
| $a$      | 29                            | 1              | $a$                            | 14                                      |
| $a^2$    | 14                            | 0              | Unchanged                      | 7                                       |
| $a^4$    | 7                             | 1              | $a \cdot a^4 = a^5$            | 3                                       |
| $a^8$    | 3                             | 1              | $a^5 \cdot a^8 = a^{13}$       | 1                                       |
| $a^{16}$ | 1                             | 1              | $a^{13} \cdot a^{16} = a^{29}$ | 0                                       |

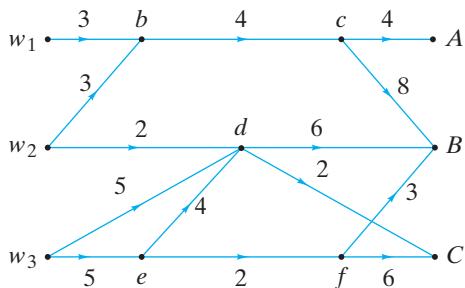
Figure 5.2.4 Computing  $a^{29}$  using repeated squaring.

We square  $x$  to obtain  $a^2$ . We then compute  $n \bmod 2$ . Since this value is 0, we know that  $2 = 2^1$  is not included in the binary expansion of 29. Therefore  $a^2$  is not included in the product, and *Result* is unchanged. We then compute the quotient when 14 is divided by 2. The quotient 7 becomes the new value of  $n$ . We then repeat this process.

3.



4. The following graph represents a pumping network in which oil for three refineries,  $A$ ,  $B$ , and  $C$ , is delivered from three wells,  $w_1$ ,  $w_2$ , and  $w_3$ . The capacities of the intermediate systems are shown on the edges. Vertices  $b$ ,  $c$ ,  $d$ ,  $e$ , and  $f$  represent intermediate pumping stations. Model this system as a network.



5. Model the system of Exercise 4 as a network assuming that well  $w_1$  can pump at most 2 units, well  $w_2$  at most 4 units, and well  $w_3$  at most 7 units.
6. Model the system of Exercise 5 as a network assuming, in addition to the limitations on the wells, that city  $A$  requires 4 units, city  $B$  requires 3 units, and city  $C$  requires 4 units.
7. Model the system of Exercise 6 as a network assuming, in addition to the limitations on the wells and the requirements by the cities, that the intermediate pumping station  $d$  can pump at most 6 units.

8. There are two routes from city  $A$  to city  $D$ . One route passes through city  $B$  and the other route passes through city  $C$ . During the period 7:00 A.M. to 8:00 A.M., the average trip times are

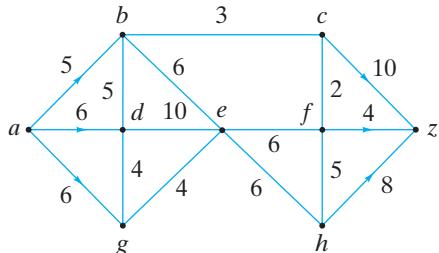
|            |             |
|------------|-------------|
| $A$ to $B$ | 30 minutes  |
| $A$ to $C$ | 15 minutes  |
| $B$ to $D$ | 15 minutes  |
| $C$ to $D$ | 15 minutes. |

The maximum capacities of the routes are

|            |                |
|------------|----------------|
| $A$ to $B$ | 1000 vehicles  |
| $A$ to $C$ | 3000 vehicles  |
| $B$ to $D$ | 4000 vehicles  |
| $C$ to $D$ | 2000 vehicles. |

Represent the flow of traffic from  $A$  to  $D$  during the period 7:00 A.M. to 8:00 A.M. as a network.

9. In the system shown, we want to maximize the flow from  $a$  to  $z$ . The capacities are shown on the edges. The flow between two vertices, neither of which is  $a$  or  $z$ , can be in either direction. Model this system as a network.



10. Give an example of a network with exactly two maximal flows, where each  $F_{ij}$  is a nonnegative integer.
11. What is the maximum number of edges that an  $n$ -vertex network can have?

## 10.2 A Maximal Flow Algorithm

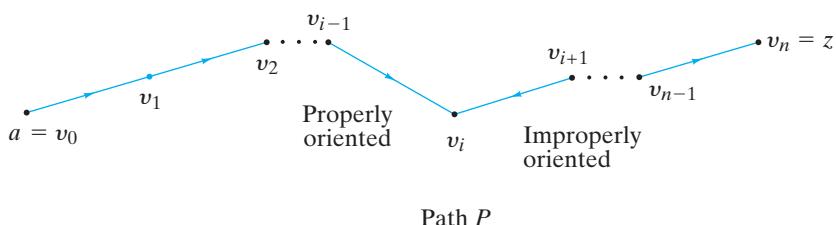
If  $G$  is a transport network, a **maximal flow** in  $G$  is a flow with maximum value. In general, there will be several flows having the same maximum value. In this section we give an algorithm for finding a maximal flow. The basic idea is simple—start with some initial flow and iteratively increase the value of the flow until no more improvement is possible. The resulting flow will then be a maximal flow.

We can take the initial flow to be the one in which the flow in each edge is zero. To increase the value of a given flow, we must find a path from the source to the sink and increase the flow along this path.

It is helpful at this point to introduce some terminology. Throughout this section,  $G$  denotes a network with source  $a$ , sink  $z$ , and capacity  $C$ . Momentarily, consider the edges of  $G$  to be undirected and let

$$P = (v_0, v_1, \dots, v_n), \quad v_0 = a, \quad v_n = z,$$

be a path from  $a$  to  $z$  in this undirected graph. (All paths in this section are with reference to the underlying undirected graph.) If an edge  $e$  in  $P$  is directed from  $v_{i-1}$  to  $v_i$ , we say that  $e$  is **properly oriented (with respect to  $P$ )**; otherwise, we say that  $e$  is **improperly oriented (with respect to  $P$ )** (see Figure 10.2.1).

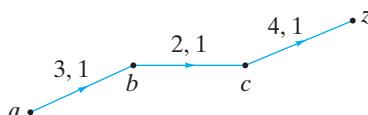


**Figure 10.2.1** Properly and improperly oriented edges. Edge  $(v_{i-1}, v_i)$  is properly oriented because it is oriented in the  $a$ -to- $z$  direction. Edge  $(v_i, v_{i+1})$  is improperly oriented because it is *not* in the  $a$ -to- $z$  direction.

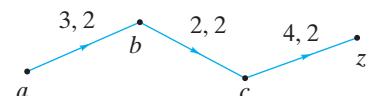
If we can find a path  $P$  from the source to the sink in which every edge in  $P$  is properly oriented and the flow in each edge is less than the capacity of the edge, it is possible to increase the value of the flow.

### Example 10.2.1

Consider the path from  $a$  to  $z$  in Figure 10.2.2. All the edges in  $P$  are properly oriented. The value of the flow in this network can be increased by 1, as shown in Figure 10.2.3.

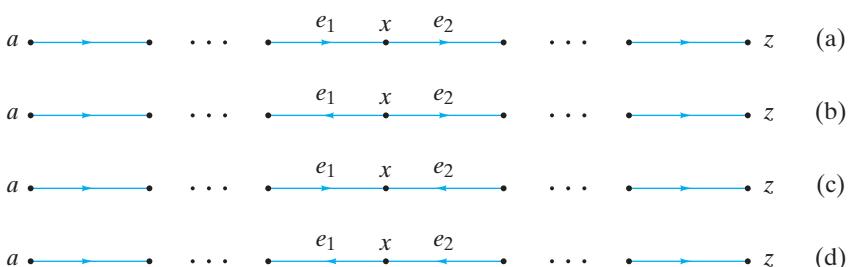


**Figure 10.2.2** A path all of whose edges are properly oriented.



**Figure 10.2.3** After increasing the flow of Figure 10.2.2 by 1.

It is also possible to increase the flow in certain paths from the source to the sink in which we have properly and improperly oriented edges. Let  $P$  be a path from  $a$  to  $z$  and let  $x$  be a vertex in  $P$  that is neither  $a$  nor  $z$  (see Figure 10.2.4). There are four possibilities for the orientations of the edges  $e_1$  and  $e_2$  incident on  $x$ . In case (a), both edges are properly oriented. In this case, if we increase the flow in each edge by  $\Delta$ , the flow into  $x$  will still equal the flow out of  $x$ . In case (b), if we increase the flow in  $e_2$  by  $\Delta$ , we must *decrease* the flow in  $e_1$  by  $\Delta$  so that the flow into  $x$  will still equal the flow out of  $x$ . Case (c) is similar to case (b), except that we increase the flow in  $e_1$  by  $\Delta$  and decrease the flow in  $e_2$  by  $\Delta$ . In case (d), we decrease the flow in both edges by  $\Delta$ . In every case, the resulting edge assignments give a flow. Of course, to carry out

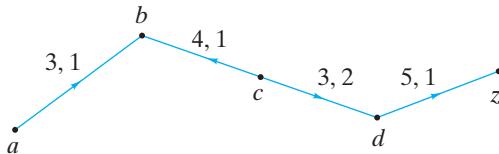


**Figure 10.2.4** The four possible orientations of the edges incident on  $x$ .

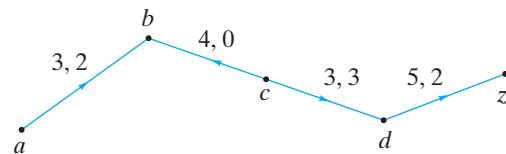
these alterations, we must have flow less than capacity in a properly oriented edge and a nonzero flow in an improperly oriented edge.

### Example 10.2.2

Consider the path from  $a$  to  $z$  in Figure 10.2.5. Edges  $(a, b)$ ,  $(c, d)$ , and  $(d, z)$  are properly oriented and edge  $(c, b)$  is improperly oriented. We decrease the flow by 1 in the improperly oriented edge  $(c, b)$  and increase the flow by 1 in the properly oriented edges  $(a, b)$ ,  $(c, d)$ , and  $(d, z)$  (see Figure 10.2.6). The value of the new flow is 1 greater than that of the original flow.



**Figure 10.2.5** A path with an improperly oriented edge:  $(c, b)$ .



**Figure 10.2.6** After increasing the flow of Figure 10.2.5 by 1.

We summarize the method of Examples 10.2.1 and 10.2.2 as a theorem.

### Theorem 10.2.3

#### Go Online

For more on the maximal flow algorithm, see [goo.gl/Zs13xb](http://goo.gl/Zs13xb)

Let  $P$  be a path from  $a$  to  $z$  in a network  $G$  satisfying the following conditions:

(a) For each properly oriented edge  $(i, j)$  in  $P$ ,

$$F_{ij} < C_{ij}.$$

(b) For each improperly oriented edge  $(i, j)$  in  $P$ ,

$$0 < F_{ij}.$$

Let

$$\Delta = \min X,$$

where  $X$  consists of the numbers  $C_{ij} - F_{ij}$ , for properly oriented edges  $(i, j)$  in  $P$ , and  $F_{ij}$ , for improperly oriented edges  $(i, j)$  in  $P$ . Define

$$F_{ij}^* = \begin{cases} F_{ij} & \text{if } (i, j) \text{ is not in } P \\ F_{ij} + \Delta & \text{if } (i, j) \text{ is properly oriented in } P \\ F_{ij} - \Delta & \text{if } (i, j) \text{ is not properly oriented in } P. \end{cases}$$

Then  $F^*$  is a flow whose value is  $\Delta$  greater than the value of  $F$ .

**Proof** (See Figures 10.2.2, 10.2.3, 10.2.5, and 10.2.6.) The argument that  $F^*$  is a flow is given just before Example 10.2.2. Since the edge  $(a, v)$  in  $P$  is increased by  $\Delta$ , the value of  $F^*$  is  $\Delta$  greater than the value of  $F$ .  $\blacktriangleleft$

In the next section we show that if there are no paths satisfying the conditions of Theorem 10.2.3, the flow is maximal. Thus it is possible to construct an algorithm based on Theorem 10.2.3. The outline is as follows:

1. Start with a flow (e.g., the flow in which the flow in each edge is 0).
2. Search for a path satisfying the conditions of Theorem 10.2.3. If no such path exists, stop; the flow is maximal.

3. Increase the flow through the path by  $\Delta$ , where  $\Delta$  is defined as in Theorem 10.2.3, and go to line 2.

In the formal algorithm, we search for a path satisfying the conditions of Theorem 10.2.3 while simultaneously keeping track of the quantities  $C_{ij} - F_{ij}$ ,  $F_{ij}$ .

### Algorithm 10.2.4

#### Finding a Maximal Flow in a Network

This algorithm finds a maximal flow in a network. The capacity of each edge is a nonnegative integer.

```

Input: A network with source a , sink z , capacity C , vertices $a = v_0, \dots, v_n = z$, and n
Output: A maximal flow F

max_flow(a, z, C, v, n) {
 // v 's label is ($predecessor(v)$, $val(v)$)
 // start with zero flow
 1. for each edge (i, j)
 2. $F_{ij} = 0$
 3. while (true) {
 // remove all labels
 4. for $i = 0$ to n {
 5. $predecessor(v_i) = null$
 6. $val(v_i) = null$
 7. }
 // label a
 8. $predecessor(a) = -$
 9. $val(a) = \infty$
 // U is the set of unexamined, labeled vertices
 10. $U = \{a\}$
 // continue until z is labeled
 11. while ($val(z) == null$) {
 12. if ($U == \emptyset$) // flow is maximal
 13. return F
 14. choose v in U
 15. $U = U - \{v\}$
 16. $\Delta = val(v)$
 17. for each edge (v, w) with $val(w) == null$
 18. if ($F_{vw} < C_{vw}$) {
 19. $predecessor(w) = v$
 20. $val(w) = \min\{\Delta, C_{vw} - F_{vw}\}$
 21. $U = U \cup \{w\}$
 22. }
 23. for each edge (w, v) with $val(w) == null$
 24. if ($F_{wv} > 0$) {
 25. $predecessor(w) = v$
 26. $val(w) = \min\{\Delta, F_{wv}\}$
 27. $U = U \cup \{w\}$
 28. }
 29. } // end while ($val(z) == null$) loop
 // find path P from a to z on which to revise flow
}

```

```

30. $w_0 = z$
31. $k = 0$
32. while ($w_k \neq a$) {
33. $w_{k+1} = predecessor(w_k)$
34. $k = k + 1$
35. }
36. $P = (w_{k+1}, w_k, \dots, w_1, w_0)$
37. $\Delta = val(z)$
38. for $i = 1$ to $k + 1$ {
39. $e = (w_i, w_{i-1})$
40. if (e is properly oriented in P)
41. $F_e = F_e + \Delta$
42. else
43. $F_e = F_e - \Delta$
44. }
45. } // end while (true) loop
 }

```

A proof that Algorithm 10.2.4 terminates is left as an exercise (Exercise 19). If the capacities are allowed to be nonnegative rational numbers, the algorithm also terminates; however, if arbitrary nonnegative real capacities are allowed and we permit the edges in line 17 to be examined in any order, the algorithm may not terminate (see [Ford, pp. 21–22]).

Algorithm 10.2.4 is often referred to as the **labeling procedure**. We will illustrate the algorithm with two examples.

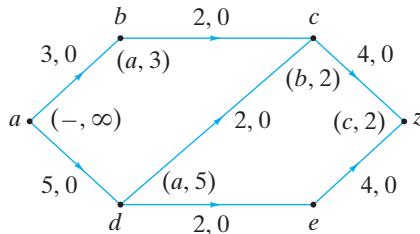
### Example 10.2.5

In this discussion, if vertex  $v$  satisfies

$$predecessor(v) = p \quad \text{and} \quad val(v) = t,$$

we show  $v$ 's label on the graph as  $(p, t)$ .

At lines 1 and 2, we initialize the flow to 0 in each edge (see Figure 10.2.7). Next, at lines 4–7 we set all labels to *null*. Then, at lines 8 and 9 we label vertex  $a$  as  $(-, \infty)$ . At line 10 we set  $U = \{a\}$ . We then enter the while loop (line 11).



**Figure 10.2.7** After the first labeling. Vertex  $v$  is labeled  $(predecessor(v), val(v))$ .

Since  $z$  is not labeled and  $U$  is not empty, we move to line 14, where we choose vertex  $a$  in  $U$  and remove it from  $U$  at line 15. At this point,  $U = \emptyset$ . We set  $\Delta$  to  $\infty [= val(a)]$  at line 16. At line 17 we examine the edges  $(a, b)$  and  $(a, d)$  since neither  $b$  nor  $d$  is labeled. For edge  $(a, b)$  we have  $F_{ab} = 0 < C_{ab} = 3$ . At lines 19 and 20, we label vertex  $b$  as  $(a, 3)$  since  $predecessor(b) = a$  and

$$val(b) = \min\{\Delta, 3 - 0\} = \min\{\infty, 3 - 0\} = 3.$$

At line 21, we add  $b$  to  $U$ . Similarly, we label vertex  $d$  as  $(a, 5)$  and add  $d$  to  $U$ . At this point,  $U = \{b, d\}$ .

We then return to the top of the while loop (line 11). Since  $z$  is not labeled and  $U$  is not empty, we move to line 14, where we choose a vertex in  $U$ . Suppose that we choose  $b$ . We remove  $b$  from  $U$  at line 15. We set  $\Delta$  to 3 [=  $val(b)$ ] at line 16. At line 17 we examine edge  $(b, c)$ . At lines 19 and 20 we label vertex  $c$  as  $(b, 2)$  since  $predecessor(c) = b$  and

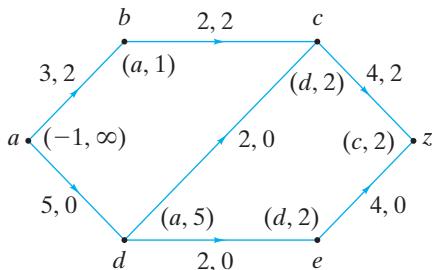
$$val(c) = \min\{\Delta, 2 - 0\} = \min\{3, 2 - 0\} = 2.$$

At line 21 we add  $c$  to  $U$ . At this point,  $U = \{c, d\}$ .

We then return to the top of the while loop (line 11). Since  $z$  is not labeled and  $U$  is not empty, we move to line 14, where we choose a vertex in  $U$ . Suppose that we choose  $c$ . We remove  $c$  from  $U$  at line 15. We set  $\Delta$  to 2 [=  $val(c)$ ] at line 16. At line 17 we examine edge  $(c, z)$ . At lines 19 and 20 we label vertex  $z$  as  $(c, 2)$ . At line 21, we add  $z$  to  $U$ . At this point,  $U = \{d, z\}$ .

We then return to the top of the while loop (line 11). Since  $z$  is labeled, we proceed to line 30. At lines 30–36, by following predecessors from  $z$ , we find the path  $P = (a, b, c, z)$  from  $a$  to  $z$ . At line 37 we set  $\Delta$  to 2. Since each edge in  $P$  is properly oriented, at line 41 we increase the flow in each edge in  $P$  by  $\Delta = 2$  to obtain Figure 10.2.8.

We then return to the top of the while loop (line 3). Next, at lines 4–7 we set all labels to *null*. Then, at lines 8 and 9 we label vertex  $a$  as  $(-, \infty)$  (see Figure 10.2.8). At line 10 we set  $U = \{a\}$ . We then enter the while loop (line 11).



**Figure 10.2.8** After increasing the flow on path  $(a, b, c, z)$  by 2 and the second labeling.

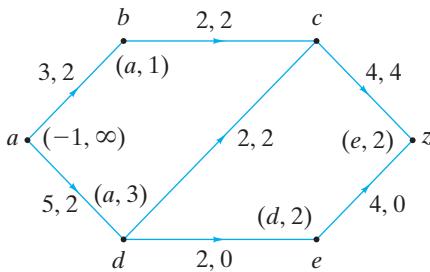
Since  $z$  is not labeled and  $U$  is not empty, we move to line 14, where we choose vertex  $a$  in  $U$  and remove it from  $U$  at line 15. At lines 19 and 20 we label vertex  $b$  as  $(a, 1)$ , and we label vertex  $d$  as  $(a, 5)$ . We add  $b$  and  $d$  to  $U$  so that  $U = \{b, d\}$ .

We then return to the top of the while loop (line 11). Since  $z$  is not labeled and  $U$  is not empty, we move to line 14, where we choose a vertex in  $U$ . Suppose that we choose  $b$ . We remove  $b$  from  $U$  at line 15. At line 17 we examine edge  $(b, c)$ . Since  $F_{bc} = C_{bc}$ , we do not label vertex  $c$  at this point. Now  $U = \{d\}$ .

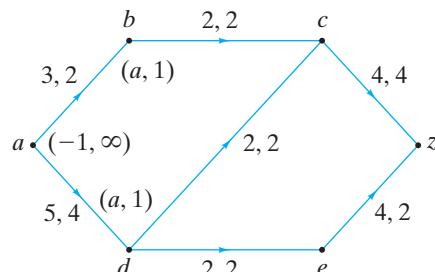
We then return to the top of the while loop (line 11). Since  $z$  is not labeled and  $U$  is not empty, we move to line 14, where we choose vertex  $d$  in  $U$  and remove it from  $U$  at line 15. At lines 19 and 20 we label vertex  $c$  as  $(d, 2)$  and we label vertex  $e$  as  $(d, 2)$ . We add  $c$  and  $e$  to  $U$  so that  $U = \{c, e\}$ .

We then return to the top of the while loop (line 11). Since  $z$  is not labeled and  $U$  is not empty, we move to line 14, where we choose a vertex in  $U$ . Suppose that we choose  $c$  in  $U$  and remove it from  $U$  at line 15. At lines 19 and 20 we label vertex  $z$  as  $(c, 2)$ . We add  $z$  to  $U$  so that  $U = \{z, e\}$ .

We then return to the top of the while loop (line 11). Since  $z$  is labeled, we proceed to line 30. At line 36 we find that  $P = (a, d, c, z)$ . Since each edge in  $P$  is properly oriented, at line 41 we increase the flow in each edge in  $P$  by  $\Delta = 2$  to obtain Figure 10.2.9.



**Figure 10.2.9** After increasing the flow on path  $(a, d, c, z)$  by 2 and the third labeling.



**Figure 10.2.10** After increasing the flow on path  $(a, d, e, z)$  by 2 and the fourth and final labeling. The flow is maximal.

You should check that the next iteration of the algorithm produces the labeling shown in Figure 10.2.9. Increasing the flow by  $\Delta = 2$  produces Figure 10.2.10.

We then return to the top of the while loop (line 3). Next, at lines 4–7 we set all labels to *null*. Then, at lines 8 and 9 we label vertex  $a$  as  $(-, \infty)$  (see Figure 10.2.10). At line 10 we set  $U = \{a\}$ . We then enter the while loop (line 11).

Since  $z$  is not labeled and  $U$  is not empty, we move to line 14, where we choose vertex  $a$  in  $U$  and remove it from  $U$  at line 15. At lines 19 and 20 we label vertex  $b$  as  $(a, 1)$  and we label vertex  $d$  as  $(a, 1)$ . We add  $b$  and  $d$  to  $U$  so that  $U = \{b, d\}$ .

We then return to the top of the while loop (line 11). Since  $z$  is not labeled and  $U$  is not empty, we move to line 14, where we choose a vertex in  $U$ . Suppose that we choose  $b$ . We remove  $b$  from  $U$  at line 15. At line 17, we examine edge  $(b, c)$ . Since  $F_{bc} = C_{bc}$ , we do not label vertex  $c$ . Now  $U = \{d\}$ .

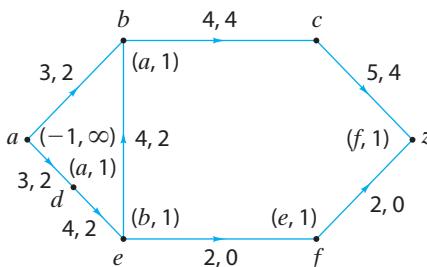
We then return to the top of the while loop (line 11). Since  $z$  is not labeled and  $U$  is not empty, we move to line 14, where we choose vertex  $d$  in  $U$  and remove it from  $U$  at line 15. At line 17 we examine edges  $(d, c)$  and  $(d, e)$ . Since  $F_{dc} = C_{dc}$  and  $F_{de} = C_{de}$ , we do not label either vertex  $c$  or vertex  $e$ . Now  $U = \emptyset$ .

We then return to the top of the while loop (line 11). Since  $z$  is not labeled, we move to line 12. Since  $U$  is empty, the algorithm terminates. The flow of Figure 10.2.10 is maximal.  $\blacktriangleleft$

Our last example shows how to modify Algorithm 10.2.4 to generate a maximal flow from a given flow.

### Example 10.2.6

Replace the zero flow in lines 1 and 2 of Algorithm 10.2.4 with the flow of Figure 10.2.11 and then find a maximal flow.



**Figure 10.2.11** After labeling.

**SOLUTION** After initializing the given flow, we move to lines 4–7, where we set all labels to *null*. Then, at lines 8 and 9 we label vertex  $a$  as  $(-, \infty)$  (see Figure 10.2.11). At line 10 we set  $U = \{a\}$ . We then enter the while loop (line 11).

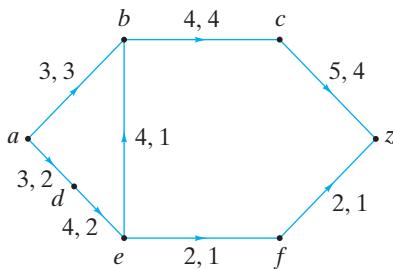
Since  $z$  is not labeled and  $U$  is not empty, we move to line 14, where we choose vertex  $a$  in  $U$  and remove it from  $U$  at line 15. At lines 19 and 20, we label vertex  $b$  as  $(a, 1)$  and we label vertex  $d$  as  $(a, 1)$ . We add  $b$  and  $d$  to  $U$  so that  $U = \{b, d\}$ .

We then return to the top of the while loop (line 11). Since  $z$  is not labeled and  $U$  is not empty, we move to line 14, where we choose a vertex in  $U$ . Suppose that we choose  $b$ . We remove  $b$  from  $U$  at line 15. At line 17 we examine edges  $(b, c)$  and  $(e, b)$ . Since  $F_{bc} = C_{bc}$ , we do not label vertex  $c$ . At lines 25 and 26, vertex  $e$  is labeled  $(b, 1)$  since

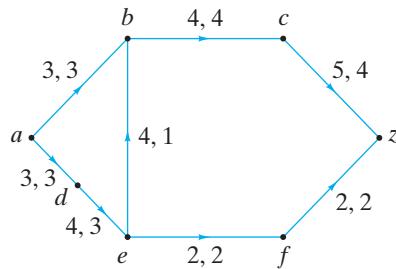
$$\text{val}(e) = \min\{\text{val}(b), F_{eb}\} = \min\{1, 2\} = 1.$$

We then return to the top of the while loop (line 11). We ultimately label  $z$  (see Figure 10.2.11), and at line 36 we find the path  $P = (a, b, e, f, z)$ . Edges  $(a, b)$ ,  $(e, f)$ , and  $(f, z)$  are properly oriented, so the flow in each is increased by 1. Since edge  $(e, b)$  is improperly oriented, its flow is decreased by 1. We obtain the flow of Figure 10.2.12.

Another iteration of the algorithm produces the maximal flow shown in Figure 10.2.13.



**Figure 10.2.12** After increasing the flow on path  $(a, b, e, f, z)$  by 1. Notice that edge  $(e, b)$  is improperly oriented and so has its flow *decreased* by 1.



**Figure 10.2.13** After increasing the flow on path  $(a, d, e, f, z)$  by 1. The flow is maximal.

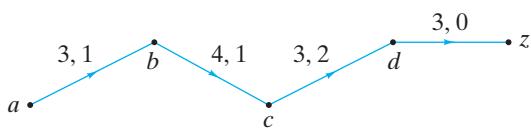
## 10.2 Review Exercises

1. What is a maximal flow?
2. What is a properly oriented edge with respect to a path?
3. What is an improperly oriented edge with respect to a path?
4. When can we increase the flow in a path from the source to the sink?
5. Explain how to increase the flow under the conditions of Exercise 4.
6. Explain how to find a maximal flow in a network.

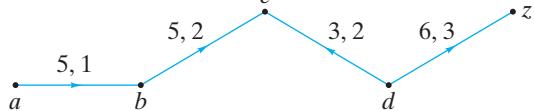
## 10.2 Exercises

In Exercises 1–3, a path from the source  $a$  to the sink  $z$  in a network is given. Find the maximum possible increase in the flow obtainable by altering the flows in the edges in the path.

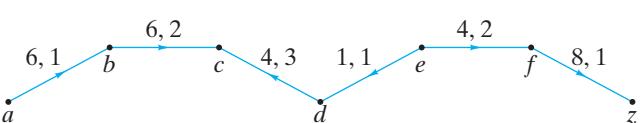
1.



2.



3.

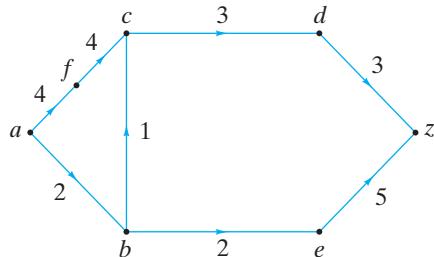


In Exercises 4–12, use Algorithm 10.2.4 to find a maximal flow in each network.

4. Figure 10.1.4

5. Figure 10.1.5

6.



7. Exercise 5, Section 10.1

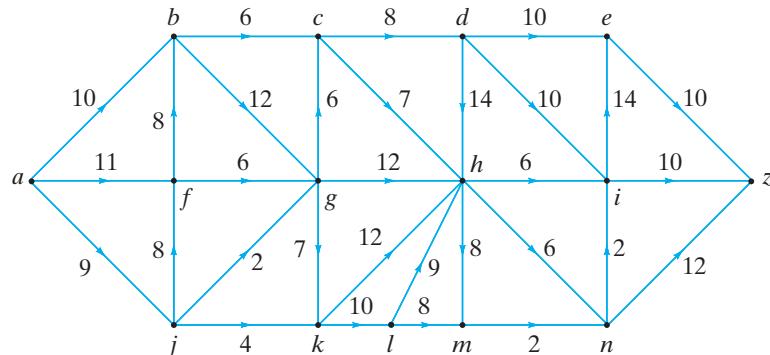
8. Exercise 6, Section 10.1

9. Exercise 7, Section 10.1

10. Exercise 8, Section 10.1

11. Exercise 9, Section 10.1

12.



In Exercises 13–18, find a maximal flow in each network starting with the flow given.

13. Figure 10.1.2

15. Exercise 2, Section 10.1

17. Figure 10.1.4 with flows

$$\begin{array}{llll} F_{a,w_1} = 2, & F_{w_1,b} = 2, & F_{bA} = 0, & F_{cA} = 0, \\ F_{Az} = 0, & F_{a,w_2} = 0, & F_{w_2,b} = 0, & F_{bc} = 2, \\ F_{cB} = 4, & F_{Bz} = 4, & F_{a,w_3} = 2, & F_{w_3,d} = 2, \\ F_{dc} = 2. & & & \end{array}$$

18. Figure 10.1.4 with flows

$$\begin{array}{llll} F_{a,w_1} = 1, & F_{w_1,b} = 1, & F_{bA} = 4, & \\ F_{cA} = 2, & F_{Az} = 6, & F_{a,w_2} = 3, & \\ F_{w_2,b} = 3, & F_{bc} = 0, & F_{cB} = 1, & \\ F_{Bz} = 1, & F_{a,w_3} = 3, & F_{w_3,d} = 3, & \\ F_{dc} = 3. & & & \end{array}$$

19. Show that Algorithm 10.2.4 terminates.

## 10.3 The Max Flow, Min Cut Theorem

In this section we show that at the termination of Algorithm 10.2.4, the flow in the network is maximal. Along the way we will define and discuss cuts in networks.

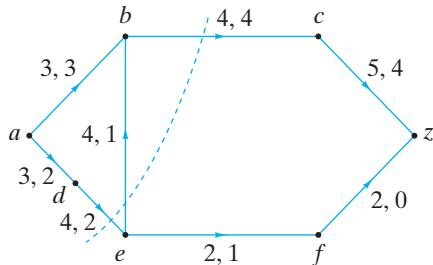
Let  $G$  be a network and consider the flow  $F$  at the termination of Algorithm 10.2.4. Some vertices are labeled and some are unlabeled. Let  $P$  ( $\bar{P}$ ) denote the set of labeled (unlabeled) vertices. (Recall that  $\bar{P}$  denotes the complement of  $P$ .) Then the source  $a$  is in  $P$  and the sink  $z$  is in  $\bar{P}$ . The set  $S$  of edges  $(v, w)$ , with  $v \in P$  and  $w \in \bar{P}$ , is called a **cut**, and the sum of the capacities of the edges in  $S$  is called the **capacity of the cut**. We will see that this cut has a minimum capacity and, since a minimal cut corresponds to a maximal flow (Theorem 10.3.9), the flow  $F$  is maximal. We begin with the formal definition of cut.

Throughout this section,  $G$  is a network with source  $a$  and sink  $z$ . The capacity of edge  $(i, j)$  is  $C_{ij}$ .

**Definition 10.3.1** ► A cut  $(P, \bar{P})$  in  $G$  consists of a set  $P$  of vertices and the complement  $\bar{P}$  of  $P$ , with  $a \in P$  and  $z \in \bar{P}$ .

### Example 10.3.2

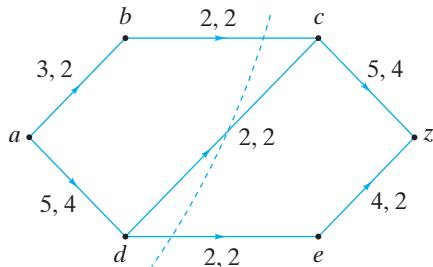
Consider the network  $G$  of Figure 10.3.1. If we let  $P = \{a, b, d\}$ , then  $\bar{P} = \{c, e, f, z\}$  and  $(P, \bar{P})$  is a cut in  $G$ . As shown, we sometimes indicate a cut by drawing a dashed line to partition the vertices.



**Figure 10.3.1** A cut in a network. The dashed line divides the vertices into sets  $P = \{a, b, d\}$  and  $\bar{P} = \{c, e, f, z\}$  producing the cut  $(P, \bar{P})$ .

### Example 10.3.3

Figure 10.2.10 shows the labeling at the termination of Algorithm 10.2.4 for a particular network. If we let  $P$  ( $\bar{P}$ ) denote the set of labeled (unlabeled) vertices, we obtain the cut shown in Figure 10.3.2.



**Figure 10.3.2** A network at termination of Algorithm 10.2.4. The cut  $(P, \bar{P})$ ,  $P = \{a, b, d\}$ , is obtained by letting  $P$  be the set of labeled vertices.

We next define the capacity of a cut.

**Definition 10.3.4** ► The *capacity of the cut*  $(P, \bar{P})$  is the number

$$C(P, \bar{P}) = \sum_{i \in P} \sum_{j \in \bar{P}} C_{ij}.$$

### Example 10.3.5

The capacity of the cut of Figure 10.3.1 is  $C_{bc} + C_{de} = 8$ .

### Example 10.3.6

The capacity of the cut of Figure 10.3.2 is  $C_{bc} + C_{dc} + C_{de} = 6$ .

The next theorem shows that the capacity of any cut is always greater than or equal to the value of any flow.

**Theorem 10.3.7**

Let  $F$  be a flow in  $G$  and let  $(P, \bar{P})$  be a cut in  $G$ . Then the capacity of  $(P, \bar{P})$  is greater than or equal to the value of  $F$ ; that is,

$$\sum_{i \in P} \sum_{j \in \bar{P}} C_{ij} \geq \sum_i F_{ai}. \quad (10.3.1)$$

(The notation  $\sum_i$  means the sum over all vertices  $i$ .)

**Proof** Note that

$$\sum_{j \in P} \sum_{i \in P} F_{ji} = \sum_{j \in P} \sum_{i \in P} F_{ij},$$

since either side of the equation is merely the sum of  $F_{ij}$  over all  $i, j \in P$ .

Now

$$\begin{aligned} \sum_i F_{ai} &= \sum_{j \in P} \sum_i F_{ji} - \sum_{j \in P} \sum_i F_{ij} \\ &= \sum_{j \in P} \sum_{i \in P} F_{ji} + \sum_{j \in P} \sum_{i \in \bar{P}} F_{ji} - \sum_{j \in P} \sum_{i \in P} F_{ij} - \sum_{j \in P} \sum_{i \in \bar{P}} F_{ij} \\ &= \sum_{j \in P} \sum_{i \in \bar{P}} F_{ji} - \sum_{j \in P} \sum_{i \in \bar{P}} F_{ij} \leq \sum_{j \in P} \sum_{i \in \bar{P}} F_{ji} \leq \sum_{j \in P} \sum_{i \in \bar{P}} C_{ji}. \end{aligned}$$

**Example 10.3.8**

In Figure 10.3.1, the value 5 of the flow is less than the capacity 8 of the cut.

A **minimal cut** is a cut having minimum capacity.

**Theorem 10.3.9****Max Flow, Min Cut Theorem**

Let  $F$  be a flow in  $G$  and let  $(P, \bar{P})$  be a cut in  $G$ . If equality holds in (10.3.1), then the flow is maximal and the cut is minimal. Moreover, equality holds in (10.3.1) if and only if

$$(a) F_{ij} = C_{ij} \quad \text{for } i \in P, j \in \bar{P}$$

and

$$(b) F_{ij} = 0 \quad \text{for } i \in \bar{P}, j \in P.$$

**Proof** The first statement follows immediately.

The proof of Theorem 10.3.7 shows that equality holds precisely when

$$\sum_{j \in P} \sum_{i \in \bar{P}} F_{ij} = 0 \quad \text{and} \quad \sum_{j \in P} \sum_{i \in \bar{P}} F_{ji} = \sum_{j \in P} \sum_{i \in \bar{P}} C_{ji};$$

thus the last statement is also true.

**Example 10.3.10**

In Figure 10.3.2, the value of the flow and the capacity of the cut are both 6; therefore, the flow is maximal and the cut is minimal.

We can use Theorem 10.3.9 to show that Algorithm 10.2.4 produces a maximal flow.

**Theorem 10.3.11**

At termination, Algorithm 10.2.4 produces a maximal flow. Moreover, if  $P$  (respectively,  $\bar{P}$ ) is the set of labeled (respectively, unlabeled) vertices at the termination of Algorithm 10.2.4, the cut  $(P, \bar{P})$  is minimal.

**Proof** Let  $P$  ( $\bar{P}$ ) be the set of labeled (unlabeled) vertices of  $G$  at the termination of Algorithm 10.2.4. Consider an edge  $(i, j)$ , where  $i \in P, j \in \bar{P}$ . Since  $i$  is labeled, we must have

$$F_{ij} = C_{ij};$$

otherwise, we would have labeled  $j$  at lines 19 and 20. Now consider an edge  $(j, i)$ , where  $j \in \bar{P}, i \in P$ . Since  $i$  is labeled, we must have

$$F_{ji} = 0;$$

otherwise, we would have labeled  $j$  at lines 25 and 26. By Theorem 10.3.9, the flow at the termination of Algorithm 10.2.4 is maximal and the cut  $(P, \bar{P})$  is minimal. ◀

## 10.3 Review Exercises

1. What is a cut in a network?
2. What is the capacity of a cut?
3. What is the relation between the capacity of any cut and the value of any flow?
4. What is a minimal cut?
5. State the max flow, min cut theorem.
6. Explain how the max flow, min cut theorem proves that the algorithm of Section 10.2 correctly finds a maximal flow in a network.

## 10.3 Exercises

In Exercises 1–3, find the capacity of the cut  $(P, \bar{P})$ . Also, determine whether the cut is minimal.

1.  $P = \{a, d\}$  for Exercise 1, Section 10.1
2.  $P = \{a, d, e\}$  for Exercise 2, Section 10.1
3.  $P = \{a, b, c, d\}$  for Exercise 3, Section 10.1

In Exercises 4–16, find a minimal cut in each network.

4. Figure 10.1.1
5. Figure 10.1.4
6. Figure 10.1.5
7. Exercise 1, Section 10.1
8. Exercise 2, Section 10.1
9. Exercise 3, Section 10.1
10. Exercise 4, Section 10.1
11. Exercise 5, Section 10.1
12. Exercise 6, Section 10.1
13. Exercise 7, Section 10.1
14. Exercise 8, Section 10.1
15. Exercise 9, Section 10.1
16. Exercise 12, Section 10.2

Exercises 17–22 refer to a network  $G$  that, in addition to having nonnegative integer capacities  $C_{ij}$ , has nonnegative integer minimal edge flow requirements  $m_{ij}$ . That is, a flow  $F$  must satisfy

$$m_{ij} \leq F_{ij} \leq C_{ij}$$

for all edges  $(i, j)$ .

17. Give an example of a network  $G$ , in which  $m_{ij} \leq C_{ij}$  for all edges  $(i, j)$ , for which no flow exists.

Define

$$\begin{aligned} C(\bar{P}, P) &= \sum_{i \in \bar{P}} \sum_{j \in P} C_{ij}, \\ m(P, \bar{P}) &= \sum_{i \in P} \sum_{j \in \bar{P}} m_{ij}, & m(\bar{P}, P) &= \sum_{i \in \bar{P}} \sum_{j \in P} m_{ij}. \end{aligned}$$

18. Show that the value  $V$  of any flow satisfies  $m(P, \bar{P}) - C(\bar{P}, P) \leq V \leq C(P, \bar{P}) - m(\bar{P}, P)$  for any cut  $(P, \bar{P})$ .
19. Show that if a flow exists in  $G$ , a maximal flow exists in  $G$  with value  $\min\{C(P, \bar{P}) - m(\bar{P}, P) \mid (P, \bar{P}) \text{ is a cut in } G\}$ .
20. Assume that  $G$  has a flow  $F$ . Develop an algorithm for finding a maximal flow in  $G$ .
21. Show that if a flow exists in  $G$ , a minimal flow exists in  $G$  with value  $\max\{m(P, \bar{P}) - C(\bar{P}, P) \mid (P, \bar{P}) \text{ is a cut in } G\}$ .
22. Assume that  $G$  has a flow  $F$ . Develop an algorithm for finding a minimal flow in  $G$ .
23. True or false? If  $F$  is a flow in a network  $G$  and  $(P, \bar{P})$  is a cut in  $G$  and the capacity of  $(P, \bar{P})$  exceeds the value of the flow,  $F$ , then the cut  $(P, \bar{P})$  is not minimal and the flow  $F$  is not maximal. If true, prove it; otherwise, give a counterexample.

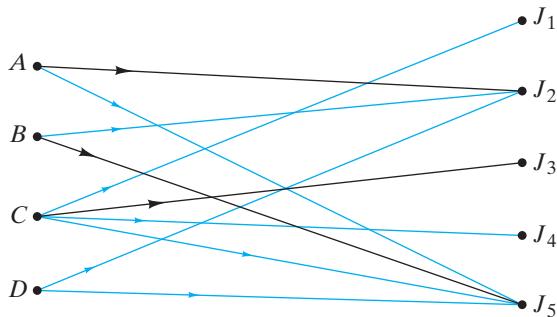
## 10.4 Matching

In this section we consider the problem of matching elements in one set to elements in another set. We will see that this problem can be reduced to finding a maximal flow in a network. We begin with an example.

### Example 10.4.1

Suppose that four persons  $A, B, C$ , and  $D$  apply for five jobs  $J_1, J_2, J_3, J_4$ , and  $J_5$ . Suppose that applicant  $A$  is qualified for jobs  $J_2$  and  $J_5$ ; applicant  $B$  is qualified for jobs  $J_2$ , and  $J_5$ ; applicant  $C$  is qualified for jobs  $J_1, J_3, J_4$ , and  $J_5$ ; and applicant  $D$  is qualified for jobs  $J_2$  and  $J_5$ . Is it possible to find a job for each applicant?

The situation can be modeled by the graph of Figure 10.4.1. The vertices represent the applicants and the jobs. An edge connects an applicant to a job for which the applicant is qualified. We can show that it is not possible to match a job to each applicant by considering applicants  $A, B$ , and  $D$ , who are qualified for jobs  $J_2$  and  $J_5$ . If  $A$  and  $B$  are assigned a job, none remains for  $D$ . Therefore, no assignments exist for  $A, B, C$ , and  $D$ .



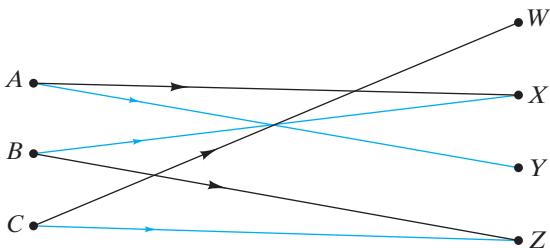
**Figure 10.4.1** Applicants ( $A, B, C, D$ ) and jobs ( $J_1, J_2, J_3, J_4, J_5$ ). An edge connects an applicant to a job for which the applicant is qualified. The black lines show a maximal matching (i.e., the maximum number of applicants have jobs). ▲

In Example 10.4.1 a matching consists of finding jobs for qualified persons. A maximal matching finds jobs for the maximum number of persons. A maximal matching for the graph of Figure 10.4.1 is shown with black lines. A complete matching finds jobs for everyone. We showed that the graph of Figure 10.4.1 has no complete matching. The formal definitions follow.

**Definition 10.4.2** ► Let  $G$  be a directed, bipartite graph with disjoint vertex sets  $V$  and  $W$  in which the edges are directed from vertices in  $V$  to vertices in  $W$ . (Any vertex in  $G$  is either in  $V$  or in  $W$ .) A *matching* for  $G$  is a set of edges  $E$  with no vertices in common. A *maximal matching* for  $G$  is a matching  $E$  in which  $E$  contains the maximum number of edges. A *complete matching* for  $G$  is a matching  $E$  having the property that if  $v \in V$ , then  $(v, w) \in E$  for some  $w \in W$ . ▲

### Example 10.4.3

The matching for the graph of Figure 10.4.2, shown with black lines, is a maximal matching and a complete matching.



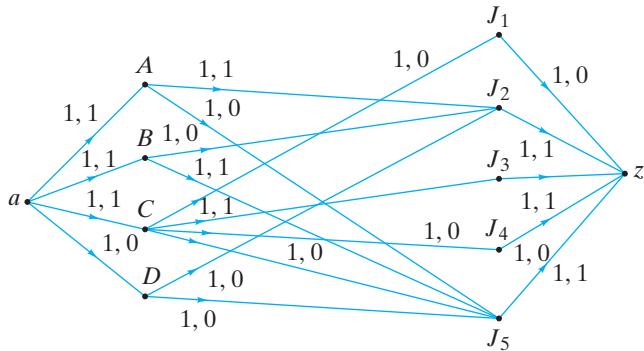
**Figure 10.4.2** The black lines show a maximal matching (the maximum number of edges are used) and a complete matching (each of  $A$ ,  $B$ , and  $C$  is matched).

In the next example we illustrate how the matching problem can be modeled as a network problem.

#### Example 10.4.4

**A Matching Network** Model the matching problem of Example 10.4.1 as a network.

**SOLUTION** We first assign each edge in the graph of Figure 10.4.1 capacity 1 (see Figure 10.4.3). Next we add a supersource  $a$  and edges of capacity 1 from  $a$  to each of  $A$ ,  $B$ ,  $C$ , and  $D$ . Finally, we introduce a supersink  $z$  and edges of capacity 1 from each of  $J_1, J_2, J_3, J_4$ , and  $J_5$  to  $z$ . We call a network such as that of Figure 10.4.3 a **matching network**.



**Figure 10.4.3** The matching problem (Figure 10.4.1) as a matching network.

The next theorem relates matching networks and flows.

#### Theorem 10.4.5

Let  $G$  be a directed, bipartite graph with disjoint vertex sets  $V$  and  $W$  in which the edges are directed from vertices in  $V$  to vertices in  $W$ . (Any vertex in  $G$  is either in  $V$  or in  $W$ .)

- (a) A flow in the matching network gives a matching in  $G$ . The vertex  $v \in V$  is matched with the vertex  $w \in W$  if and only if the flow in edge  $(v, w)$  is 1.
- (b) A maximal flow corresponds to a maximal matching.
- (c) A flow whose value is  $|V|$  corresponds to a complete matching.

**Proof** Let  $a$  ( $z$ ) represent the source (sink) in the matching network, and suppose that a flow is given.

Suppose that the edge  $(v, w)$ ,  $v \in V$ ,  $w \in W$ , has flow 1. The only edge into vertex  $v$  is  $(a, v)$ . This edge must have flow 1; thus the flow into vertex  $v$  is 1. Since the flow out of  $v$  is also 1, the only edge of the form  $(v, x)$  having flow 1 is  $(v, w)$ . Similarly, the only edge of the form  $(x, w)$  having flow 1 is  $(v, w)$ . Therefore, if  $E$  is the set of edges of the form  $(v, w)$  having flow 1, the members of  $E$  have no vertices in common; thus  $E$  is a matching for  $G$ .

Parts (b) and (c) follow from the fact that the number of vertices in  $V$  matched is equal to the value of the corresponding flow. ◀

Since a maximal flow gives a maximal matching, Algorithm 10.2.4 applied to a matching network produces a maximal matching. In practice, the implementation of Algorithm 10.2.4 can be simplified by using the adjacency matrix of the graph (see Exercise 11).

### Example 10.4.6

The matching of Figure 10.4.1 is represented as a flow in Figure 10.4.3. Since the flow is maximal, the matching is maximal. ◀

Next, we turn to the existence of a complete matching in a directed, bipartite graph  $G$  with vertex sets  $V$  and  $W$ . If  $S \subseteq V$ , we let

$$R(S) = \{w \in W \mid v \in S \text{ and } (v, w) \text{ is an edge in } G\}.$$

### Go Online

For more on the Marriage Problem, see  
[goo.gl/Zs13xb](http://goo.gl/Zs13xb)

### Theorem 10.4.7

#### Hall's Marriage Theorem

Let  $G$  be a directed, bipartite graph with disjoint vertex sets  $V$  and  $W$  in which the edges are directed from vertices in  $V$  to vertices in  $W$ . (Any vertex in  $G$  is either in  $V$  or in  $W$ .) There exists a complete matching in  $G$  if and only if

$$|S| \leq |R(S)| \quad \text{for all } S \subseteq V. \tag{10.4.1}$$

**Proof** We have already pointed out that if there is a complete matching in  $G$ , condition (10.4.1) holds.

Suppose that condition (10.4.1) holds. Let  $n = |V|$  and let  $(P, \bar{P})$  be a minimal cut in the matching network. If we can show that the capacity of this cut is  $n$ , a maximal flow would have value  $n$ . The matching corresponding to this maximal flow would be a complete matching.

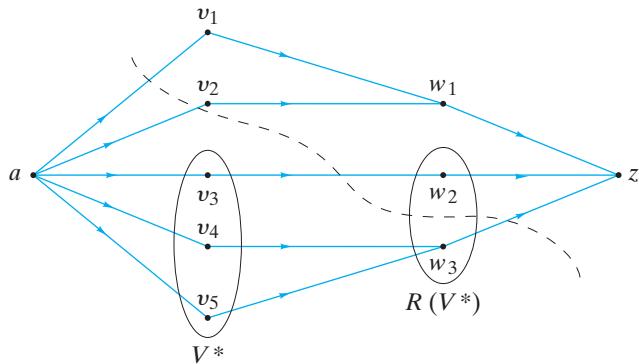
The argument is by contradiction. Assume that the capacity of the minimal cut  $(P, \bar{P})$  is less than  $n$ . The capacity of this cut is the number of edges in the set

$$E = \{(x, y) \mid x \in P, y \in \bar{P}\}$$

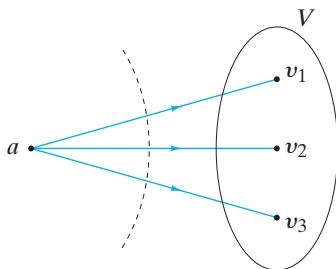
(see Figure 10.4.4). A member of  $E$  is one of the three types:

- Type I:  $(a, v)$ ,  $v \in V$ ;
- Type II:  $(v, w)$ ,  $v \in V$ ,  $w \in W$ ;
- Type III:  $(w, z)$ ,  $w \in W$ .

We will estimate the number of edges of each type.

**Figure 10.4.4** The proof of Theorem 10.4.7.

$V = \{v_1, v_2, v_3, v_4, v_5\}$ ;  $n = |V| = 5$ ;  $W = \{w_1, w_2, w_3\}$ ; the cut is  $(P, \bar{P})$ , where  $P = \{a, v_3, v_4, v_5, w_3\}$ ;  $V^* = V \cap P = \{v_3, v_4, v_5\}$ ;  $R(V^*) = \{w_2, w_3\}$ ;  $W_1 = R(V^*) \cap P = \{w_3\}$ ;  $W_2 = R(V^*) \cap \bar{P} = \{w_2\}$ ;  $E = \{(a, v_1), (a, v_2), (v_3, w_2), (w_3, z)\}$ . The capacity of the cut is  $|E| = 4 < n$ . The type I edges are  $(a, v_1)$  and  $(a, v_2)$ . Edge  $(v_3, w_2)$  is the only type II edge, and edge  $(w_3, z)$  is the only type III edge.

**Figure 10.4.5** The proof of Theorem 10.4.7 for  $n = 3$ . If  $V \subseteq \bar{P}$ , as shown the capacity of the cut is  $n$ . Since we are assuming that the capacity of the cut is less than  $n$ , this case cannot occur. Therefore,  $V \cap P$  is nonempty.

If  $V \subseteq \bar{P}$ , the capacity of the cut is  $n$  (see Figure 10.4.5); thus  $V^* = V \cap P$  is nonempty. It follows that there are  $n - |V^*|$  edges in  $E$  of type I.

We partition  $R(V^*)$  into the sets

$$W_1 = R(V^*) \cap P \quad \text{and} \quad W_2 = R(V^*) \cap \bar{P}.$$

Then there are at least  $|W_1|$  edges in  $E$  of type III. Thus there are less than

$$n - (n - |V^*|) - |W_1| = |V^*| - |W_1|$$

edges of type II in  $E$ . Since each member of  $W_2$  contributes at most one type II edge,

$$|W_2| < |V^*| - |W_1|.$$

Thus

$$|R(V^*)| = |W_1| + |W_2| < |V^*|,$$

which contradicts (10.4.1). Therefore, a complete matching exists. ◀

#### Example 10.4.8

For the graph in Figure 10.4.1, if  $S = \{A, B, D\}$ , we have  $R(S) = \{J_2, J_5\}$  and

$$|S| = 3 > 2 = |R(S)|.$$

By Theorem 10.4.7, there is not a complete matching for the graph of Figure 10.4.1. ◀

#### Example 10.4.9

There are  $n$  computers and  $n$  disk drives. Each computer is compatible with  $m > 0$  disk drives and each disk drive is compatible with  $m$  computers. Is it possible to match each computer with a compatible disk drive?

**SOLUTION** Let  $V$  be the set of computers and  $W$  be the set of disk drives. An edge exists from  $v \in V$  to  $w \in W$  if  $v$  and  $w$  are compatible. Notice that every vertex has

degree  $m$ . Let  $S = \{v_1, \dots, v_k\}$  be a subset of  $V$ . Then there are  $km$  edges from the set  $S$ . If  $R(S) = \{w_1, \dots, w_j\}$ , then  $R(S)$  receives at most  $jm$  edges from  $S$ . Therefore,  $km \leq jm$ . Now  $|S| = k \leq j = |R(S)|$ . By Theorem 10.4.7 there is a complete matching. Thus it is possible to match each computer with a compatible disk drive. ▶

## 10.4 Review Exercises

1. What is a matching?
2. What is a maximal matching?
3. What is a complete matching?
4. What is the relation between flows and matchings?
5. State Hall's Marriage Theorem.

## 10.4 Exercises

1. Show that the flow in Figure 10.4.3 is maximal by exhibiting a minimal cut whose capacity is 3.
2. Find the flow that corresponds to the matching of Figure 10.4.2. Show that this flow is maximal by exhibiting a minimal cut whose capacity is 3.

*Exercises 3–7 refer to Figure 10.4.1, where we reverse the direction of the edges so that edges are directed from jobs to applicants.*

3. What does a matching represent?
4. What does a maximal matching represent?
5. Show a maximal matching.
6. What does a complete matching represent?
7. Is there a complete matching? If there is a complete matching, show one. If there is not a complete matching, explain why none exists.
8. Applicant  $A$  is qualified for jobs  $J_1$  and  $J_4$ ;  $B$  is qualified for jobs  $J_2$ ,  $J_3$ , and  $J_6$ ;  $C$  is qualified for jobs  $J_1$ ,  $J_3$ ,  $J_5$ , and  $J_6$ ;  $D$  is qualified for jobs  $J_1$ ,  $J_3$ , and  $J_4$ ; and  $E$  is qualified for jobs  $J_1$ ,  $J_3$ , and  $J_6$ .

- (a) Model this situation as a matching network.
- (b) Use Algorithm 10.2.4 to find a maximal matching.
- (c) Is there a complete matching?

9. Applicant  $A$  is qualified for jobs  $J_1$ ,  $J_2$ ,  $J_4$ , and  $J_5$ ;  $B$  is qualified for jobs  $J_1$ ,  $J_4$ , and  $J_5$ ;  $C$  is qualified for jobs  $J_1$ ,  $J_4$ , and  $J_5$ ;  $D$  is qualified for jobs  $J_1$  and  $J_5$ ;  $E$  is qualified for jobs  $J_2$ ,  $J_3$ , and  $J_5$ ; and  $F$  is qualified for jobs  $J_4$  and  $J_5$ . Answer parts (a)–(c) of Exercise 8 for this situation.

10. Applicant  $A$  is qualified for jobs  $J_1$ ,  $J_2$ , and  $J_4$ ;  $B$  is qualified for jobs  $J_3$ ,  $J_4$ ,  $J_5$ , and  $J_6$ ;  $C$  is qualified for jobs  $J_1$  and  $J_5$ ;  $D$  is qualified for jobs  $J_1$ ,  $J_3$ ,  $J_4$ , and  $J_8$ ;  $E$  is qualified for jobs  $J_1$ ,  $J_2$ ,  $J_4$ ,  $J_6$ , and  $J_8$ ;  $F$  is qualified for jobs  $J_4$  and  $J_6$ ; and  $G$  is qualified for jobs  $J_3$ ,  $J_5$ , and  $J_7$ . Answer parts (a)–(c) of Exercise 8 for this situation.
11. Five students,  $V$ ,  $W$ ,  $X$ ,  $Y$ , and  $Z$ , are members of four committees,  $C_1$ ,  $C_2$ ,  $C_3$ , and  $C_4$ . The members of  $C_1$  are  $V$ ,  $X$ , and  $Y$ ; the members of  $C_2$  are  $X$  and  $Z$ ; the members of  $C_3$  are

$V$ ,  $Y$ , and  $Z$ ; and the members of  $C_4$  are  $V$ ,  $W$ ,  $X$ , and  $Z$ . Each committee is to send a representative to the administration. No student can represent two committees.

- (a) Model this situation as a matching network.
- (b) What is the interpretation of a maximal matching?
- (c) What is the interpretation of a complete matching?
- (d) Use Algorithm 10.2.4 to find a maximal matching.
- (e) Is there a complete matching?

12. Show that by a suitable ordering of the vertices, the adjacency matrix of a bipartite graph can be written

$$\begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix},$$

where  $0$  is a matrix consisting only of  $0$ 's and  $A^T$  is the transpose of the matrix  $A$ .

*In Exercises 13–15,  $G$  is a bipartite graph,  $A$  is the matrix of Exercise 12, and  $F$  is a flow in the associated matching network. Label each entry in  $A$  that represents an edge with flow  $f$ .*

13. What kind of labeling corresponds to a matching?
14. What kind of labeling corresponds to a complete matching?
15. What kind of labeling corresponds to a maximal matching?
16. Restate Algorithm 10.2.4, applied to a matching network, in terms of operations on the matrix  $A$  of Exercise 12.

*Let  $G$  be a directed, bipartite graph with disjoint vertex sets  $V$  and  $W$  in which the edges are directed from vertices in  $V$  to vertices in  $W$ . (Any vertex in  $G$  is either in  $V$  or in  $W$ .) We define the deficiency of  $G$  as*

$$\delta(G) = \max\{|S| - |R(S)| \mid S \subseteq V\}.$$

17. Show that  $G$  has a complete matching if and only if  $\delta(G) = 0$ .
18. Show that the maximum number of vertices in  $V$  that can be matched with vertices in  $W$  is  $|V| - \delta(G)$ .
19. True or false? Any matching is contained in a maximal matching. If true, prove it; if false, give a counterexample.

## Problem-Solving Corner

### Problem

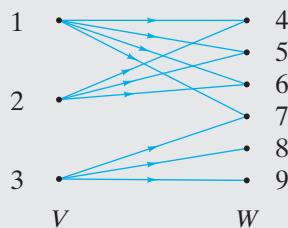
Let  $G$  be a directed, bipartite graph with disjoint vertex sets  $V$  and  $W$  in which the edges are directed from vertices in  $V$  to vertices in  $W$ . (Any vertex in  $G$  is either in  $V$  or in  $W$ .) Let  $M_W$  denote the maximum degree that occurs among vertices in  $W$ , and let  $m_V$  denote the minimum degree that occurs among vertices in  $V$ . Show that if  $0 < M_W \leq m_V$ , then  $G$  has a complete matching.

### Attacking the Problem

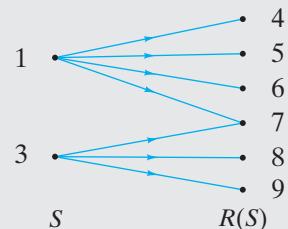
Hall's Marriage Theorem (Theorem 10.4.7) says that a directed, bipartite graph with disjoint vertex sets  $V$  and  $W$  has a complete matching if and only if  $|S| \leq |R(S)|$  for all  $S \subseteq V$ . Thus a possible way to solve the problem is to show that the given condition  $M_W \leq m_V$  implies that  $|S| \leq |R(S)|$  for all  $S \subseteq V$ .

### Finding a Solution

Our goal is to prove that if  $M_W \leq m_V$ , then  $|S| \leq |R(S)|$  for all  $S \subseteq V$ . Let's start with an example graph  $G$  for which  $M_W \leq m_V$ ; in fact, let's arrange to have  $M_W = 2$  and  $m_V = 3$ :



Consider an example subset  $S = \{1, 3\}$  of  $V$  and the edges incident on vertices in  $S$ :



The fact that the minimum degree of vertices in  $V$  is 3 means that for any subset  $S$  of  $V$ , at least three edges are incident on each vertex in  $S$ . In general, there will be at least  $3|S| = m_V|S|$  edges incident on vertices in  $S$ . In our example,  $3|S| = 6$ , but there are actually seven

## Matching

edges incident on vertices in  $S$ . The expression  $m_V|S|$  is always a *lower bound* for the number of edges incident on vertices in  $S$ .

The fact that the maximum degree of vertices in  $W$  is 2 means that for any subset  $S$  of  $V$ , at most two edges are incident on each vertex in  $R(S)$ . In general, there will be at most  $2|R(S)| = M_W|R(S)|$  edges incident on vertices in  $R(S)$ . In our example,  $2|R(S)| = 12$ , but there are actually 10 edges incident on vertices in  $R(S)$ . Since the edges incident on vertices in  $S$  are a subset of the edges incident on vertices in  $R(S)$ , the expression  $M_W|R(S)|$  is always an *upper bound* for the number of edges incident on vertices in  $S$ .

We have two ways of estimating the number of edges incident on vertices in  $S$ . The first way, using  $S$ , gives us a lower bound  $m_V|S|$  on the number of edges, and the second way, using  $R(S)$ , gives us an upper bound  $M_W|R(S)|$  on the number of edges. Comparing these estimates gives us the inequality

$$m_V|S| \leq M_W|R(S)|.$$

We can't deduce  $|S| \leq |R(S)|$ , but we haven't used the hypothesis

$$M_W \leq m_V$$

yet! Combining the last two inequalities, we have

$$m_V|S| \leq M_W|R(S)| \leq m_V|R(S)|.$$

If we now cancel  $m_V$  from both ends of the inequality, we obtain

$$|S| \leq |R(S)|,$$

which is exactly the inequality we wanted to prove.

### Formal Solution

Let  $S \subseteq V$ . Each vertex in  $S$  is incident on at least  $m_V|S|$  edges; thus there are at least  $m_V|S|$  edges incident on vertices in  $S$ . Each vertex in  $R(S)$  is incident on at most  $M_W$  edges; thus there are at most  $M_W|R(S)|$  edges incident on vertices in  $R(S)$ . It follows that  $m_V|S| \leq M_W|R(S)|$ . Since  $M_W \leq m_V$ ,  $|R(S)|M_W \leq |R(S)|m_V$ . Therefore,  $m_V|S| \leq m_V|R(S)|$ , and  $|S| \leq |R(S)|$ . By Theorem 10.4.7,  $G$  has a complete matching.

### Summary of Problem-Solving Techniques

- Look at example graphs.
- When looking at examples, it's a good idea to assign distinct values to the parameters in the

problem so you can keep them straight. (In our example we set  $M_W = 2$  and  $m_V = 3$ .)

- Try to reduce given conditions to those in a useful theorem. (We reduced the conditions given in this problem to the conditions given in Hall's Marriage Theorem.)
- An inequality can sometimes be proved by estimating the size of some set in two different ways. If one estimate gives an upper bound  $M$  and another gives a lower bound  $m$ , it follows that  $m \leq M$ .

### Comments

The last summarized problem-solving technique provides a method of proving an inequality. In a similar

way, an equality can sometimes be proved by counting the number of elements in some set in two different ways. If one way of counting gives  $c_1$  and the other way of counting gives  $c_2$ , it follows that  $c_1 = c_2$ . These techniques are widely used and their usefulness cannot be overemphasized. For example, in Section 6.2 we derived a formula for  $C(n, r)$  by counting the number of  $r$ -permutations of an  $n$ -element set in two different ways.

### Exercise

1. Give an example of a bipartite graph  $G$  that has a complete matching but does not satisfy the condition  $M_W \leq m_V$ .

## Chapter 10 Notes

General references that contain sections on network models are [Berge; Deo; Liu, 1968, 1985; and Tucker]. The classic work on networks is [Ford]; many of the results on networks, especially the early results, are due to Ford and Fulkerson, the authors of this book. [Tarjan] discusses network flow algorithms and implementation details.

See [Bacheli] for a nice direct proof of Hall's Marriage Theorem (Theorem 10.4.7) using mathematical induction.

The problem of finding a maximal flow in a network  $G$ , with source  $a$ , sink  $z$ , and capacities  $C_{ij}$ , may be rephrased as follows:

$$\text{maximize} \sum_j F_{aj}$$

subject to

$$\begin{aligned} 0 \leq F_{ij} &\leq C_{ij} && \text{for all } i, j, \\ \sum_i F_{ij} &= \sum_i F_{ji} && \text{for all } j. \end{aligned}$$

Such a problem is an example of a **linear programming problem**. In a linear programming problem, we want to maximize (or minimize) a linear expression, such as  $\sum_j F_{aj}$ , subject to linear inequality and equality constraints, such as  $0 \leq F_{ij} \leq C_{ij}$  and  $\sum_i F_{ij} = \sum_i F_{ji}$ . Although the **simplex algorithm** is normally an efficient way to solve a general linear programming problem, network transport problems are usually more efficiently solved using Algorithm 10.2.4. See [Hillier] for an exposition of the simplex algorithm.

Suppose that for each edge  $(i, j)$  in a network  $G$ ,  $c_{ij}$  represents the cost of the flow of one unit through edge  $(i, j)$ . Suppose that we want a maximal flow, with minimal cost

$$\sum_i \sum_j c_{ij} F_{ij}.$$

This problem, called the **transportation problem**, is again a linear programming problem and, as with the maximal flow problem, a specific algorithm can be used to obtain a solution that is, in general, more efficient than the simplex algorithm (see [Hillier]).

## Chapter 10 Review

### Section 10.1

1. (Transport) network
2. Source
3. Sink
4. Capacity
5. Flow in a network
6. Flow in an edge
7. Flow into a vertex
8. Flow out of a vertex
9. Conservation of flow
10. Given a flow  $F$  in a network, the flow out of the source equals the flow into the sink. This common value is called the value of the flow  $F$ .
11. Supersource
12. Supersink

### Section 10.2

13. Maximal flow
14. Properly oriented edge with respect to a path
15. Improperly oriented edge with respect to a path
16. How to increase the flow in a path from the source to the sink when:
  - (a) for each properly oriented edge the flow is less than the capacity and

- (b) each improperly oriented edge has positive flow (see Theorem 10.2.3)

17. How to find a maximal flow in a network (Algorithm 10.2.4)

### Section 10.3

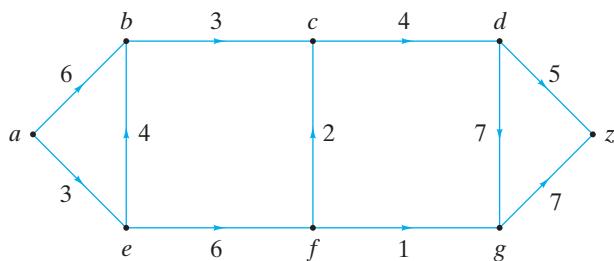
18. Cut in a network
19. Capacity of a cut
20. The capacity of any cut is greater than or equal to the value of any flow (Theorem 10.3.7).
21. Minimal cut
22. Max flow, min cut theorem (Theorem 10.3.9)
23. At the termination of the maximal flow algorithm, Algorithm 10.2.4, the set of labeled vertices defines a minimal cut.

### Section 10.4

24. Matching
25. Maximal matching
26. Complete matching
27. Matching network
28. Relationship between flows and matchings (Theorem 10.4.5)
29. Hall's Marriage Theorem (Theorem 10.4.7)

## Chapter 10 Self-Test

Exercises 1–4 refer to the following network. The capacities are shown on the edges.



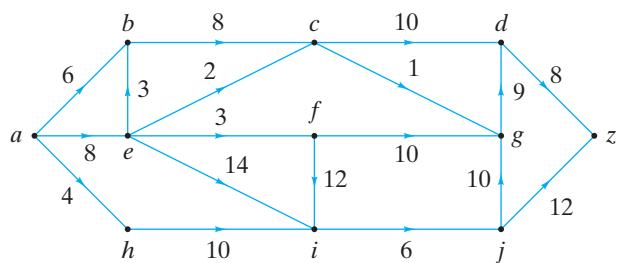
1. Explain why

$$\begin{aligned} F_{a,e} &= 2, & F_{e,b} &= 2, & F_{b,c} &= 3, \\ F_{c,d} &= 3, & F_{d,z} &= 3, & F_{a,b} &= 1, \end{aligned}$$

with all other  $F_{x,y} = 0$ , is a flow.

2. What is the flow into  $b$ ?
3. What is the flow out of  $c$ ?
4. What is the value of the flow  $F$ ?

5. For the flow of Exercise 1, find a path from  $a$  to  $z$  satisfying the following: (a) for each properly oriented edge the flow is less than the capacity and (b) each improperly oriented edge has positive flow.
6. By modifying only the flows in the edges of the path of Exercise 5, find a flow with a larger value than  $F$ .
7. Use Algorithm 10.2.4 to find a maximal flow in the network of Exercise 1 (beginning with the flow in which the flow in each edge is equal to zero).
8. Use Algorithm 10.2.4 to find a maximal flow in the following network (beginning with the flow in which the flow in each edge is equal to zero).



9. In each of parts (a)–(d), answer true if the statement is true for every network; otherwise, answer false.
- If the capacity of a cut in a network is equal to  $C_a$ , then the value of any flow is less than or equal to  $C_a$ .
  - If the capacity of a cut in a network is equal to  $C_a$ , then the value of any flow is greater than or equal to  $C_a$ .
  - If the capacity of a cut in a network is equal to  $C_a$ , then the value of some flow is greater than or equal to  $C_a$ .
  - If the capacity of a cut in a network is equal to  $C_a$ , then the value of some flow is less than or equal to  $C_a$ .
10. Find the capacity of the cut  $(P, \bar{P})$  in the network of Exercise 1, where  $P = \{a, b, e, f\}$ .
11. Is the cut  $(P, \bar{P})$ ,  $P = \{a, b, e, f\}$ , in the network of Exercise 1 minimal? Explain.
12. Find a minimal cut in the network of Exercise 8.
- Exercises 13–16 refer to the following situation. Applicant A is qualified for jobs  $J_2$ ,  $J_4$ , and  $J_5$ ; applicant B is qualified for jobs  $J_1$  and  $J_3$ ; applicant C is qualified for jobs  $J_1$ ,  $J_3$ , and  $J_5$ ; and applicant D is qualified for jobs  $J_3$  and  $J_5$ .*
- Model the situation as a matching network.
  - Use Algorithm 10.2.4 to find a maximal matching.
  - Is there a complete matching?
  - Find a minimal cut in the matching network.

## Chapter 10 Computer Exercises

- Write a program that accepts as input a network with a given flow and outputs all possible paths from the source to the sink on which the flow can be increased.
- Implement Algorithm 10.2.4 that finds a maximal flow in a network as a program. Have the program output the minimal cut as well as the maximal flow.
- Write a program that computes the deficiency of a network.



## Chapter 11

# BOOLEAN ALGEBRAS AND COMBINATORIAL CIRCUITS

- 11.1** Combinatorial Circuits
- 11.2** Properties of Combinatorial Circuits
- 11.3** Boolean Algebras
- 11.4** Boolean Functions and Synthesis of Circuits
- 11.5** Applications

### Go Online

Biographies of Boole and Shannon are at [goo.gl/5HWh0y](http://goo.gl/5HWh0y)

Several definitions honor the nineteenth-century mathematician George Boole—Boolean algebra, Boolean function, Boolean expression, and Boolean ring—to name a few. Boole is one of the persons in a long historical chain who were concerned with formalizing and mechanizing the process of logical thinking. In fact, in 1854 Boole wrote a book entitled *The Laws of Thought*. One of Boole’s contributions was the development of a theory of logic using symbols instead of words. For a discussion of Boole’s work, see [Hailperin].

Almost a century after Boole’s work, it was observed, especially by C. E. Shannon in 1938 (see [Shannon]), that Boolean algebra could be used to analyze electrical circuits. Thus Boolean algebra became an indispensable tool for the analysis and design of electronic computers in the succeeding decades. We explore the relationship of Boolean algebra to circuits throughout this chapter.

## 11.1

### Combinatorial Circuits

### Go Online

For more on combinatorial circuits, see [goo.gl/5HWh0y](http://goo.gl/5HWh0y)

In a digital computer, there are only two possibilities, written 0 and 1, for the smallest, indivisible object. All programs and data are ultimately reducible to combinations of bits. A variety of devices have been used throughout the years in digital computers to store bits. Electronic circuits allow these storage devices to communicate with each other. A bit in one part of a circuit is transmitted to another part of the circuit as a voltage. Thus two voltage levels are needed—for example, a high voltage can communicate 1 and a low voltage can communicate 0.

In this section we discuss **combinatorial circuits**. The output of a combinatorial circuit is uniquely defined for every combination of inputs. A combinatorial circuit has no memory; previous inputs and the state of the system do not affect the output of a combinatorial circuit. Circuits for which the output is a function, not only of the inputs, but also of the state of the system, are called **sequential circuits** and are considered in Chapter 12.

Combinatorial circuits can be constructed using solid-state devices, called **gates**, which are capable of switching voltage levels (bits). We will begin by discussing AND, OR, and NOT gates.

**Definition 11.1.1** ► An *AND gate* receives inputs  $x_1$  and  $x_2$ , where  $x_1$  and  $x_2$  are bits, and produces output denoted  $x_1 \wedge x_2$ , where

$$x_1 \wedge x_2 = \begin{cases} 1 & \text{if } x_1 = 1 \text{ and } x_2 = 1 \\ 0 & \text{otherwise.} \end{cases}$$

An AND gate is drawn as shown in Figure 11.1.1.



Figure 11.1.1 AND gate. ▲

**Definition 11.1.2** ► An *OR gate* receives inputs  $x_1$  and  $x_2$ , where  $x_1$  and  $x_2$  are bits, and produces output denoted  $x_1 \vee x_2$ , where

$$x_1 \vee x_2 = \begin{cases} 1 & \text{if } x_1 = 1 \text{ or } x_2 = 1 \\ 0 & \text{otherwise.} \end{cases}$$

An OR gate is drawn as shown in Figure 11.1.2.



Figure 11.1.2 OR gate. ▲

**Definition 11.1.3** ► A *NOT gate* (or *inverter*) receives input  $x$ , where  $x$  is a bit, and produces output denoted  $\bar{x}$ , where

$$\bar{x} = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } x = 1. \end{cases}$$



Figure 11.1.3 NOT gate. ▲

A NOT gate is drawn as shown in Figure 11.1.3.

The **logic table** of a combinatorial circuit lists all possible inputs together with the resulting outputs.

#### Example 11.1.4

Following are the logic tables for the basic AND, OR, and NOT circuits (Figures 11.1.1–11.1.3).

| $x_1$ | $x_2$ | $x_1 \wedge x_2$ | $x_1$ | $x_2$ | $x_1 \vee x_2$ | $x$ | $\bar{x}$ |
|-------|-------|------------------|-------|-------|----------------|-----|-----------|
| 1     | 1     | 1                | 1     | 1     | 1              | 1   | 0         |
| 1     | 0     | 0                | 1     | 0     | 1              | 0   | 1         |
| 0     | 1     | 0                | 0     | 1     | 1              |     |           |
| 0     | 0     | 0                | 0     | 0     | 0              |     |           |

We note that performing the operation AND (OR) is the same as taking the minimum (maximum) of the two bits  $x_1$  and  $x_2$ . ▲

#### Example 11.1.5

The circuit of Figure 11.1.4 is an example of a combinatorial circuit since the output  $y$  is uniquely defined for each combination of inputs  $x_1$ ,  $x_2$ , and  $x_3$ .

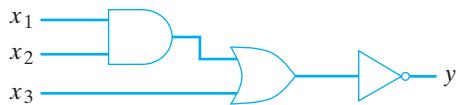
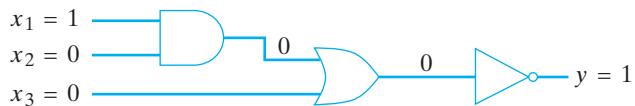


Figure 11.1.4 A combinational circuit.

The logic table for this combinational circuit follows.

| $x_1$ | $x_2$ | $x_3$ | $y$ |
|-------|-------|-------|-----|
| 1     | 1     | 1     | 0   |
| 1     | 1     | 0     | 0   |
| 1     | 0     | 1     | 0   |
| 1     | 0     | 0     | 1   |
| 0     | 1     | 1     | 0   |
| 0     | 1     | 0     | 1   |
| 0     | 0     | 1     | 0   |
| 0     | 0     | 0     | 1   |

Notice that all possible combinations of values for the inputs  $x_1$ ,  $x_2$ , and  $x_3$  are listed. For a given set of inputs, we can compute the value of the output  $y$  by tracing the flow through the circuit. For example, the fourth line of the table gives the value of the output  $y$  for the input values  $x_1 = 1$ ,  $x_2 = 0$ , and  $x_3 = 0$ . If  $x_1 = 1$  and  $x_2 = 0$ , the output from the AND gate is 0 (see Figure 11.1.5). Since  $x_3 = 0$ , the inputs to the OR gate are both 0. Therefore, the output of the OR gate is 0. Since the input to the NOT gate is 0, it produces output  $y = 1$ .

Figure 11.1.5 The circuit of Figure 11.1.4 when  $x_1 = 1$  and  $x_2 = x_3 = 0$ .**Example 11.1.6**

The circuit of Figure 11.1.6 is not a combinational circuit, because the output  $y$  is not uniquely defined for each combination of inputs  $x_1$  and  $x_2$ . For example, suppose that  $x_1 = 1$  and  $x_2 = 0$ . If the output of the AND gate is 0, then  $y = 0$ . On the other hand, if the output of the AND gate is 1, then  $y = 1$ . Such a circuit might be used to store one bit.

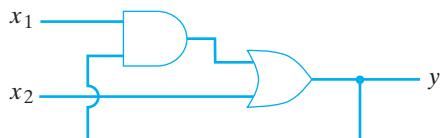
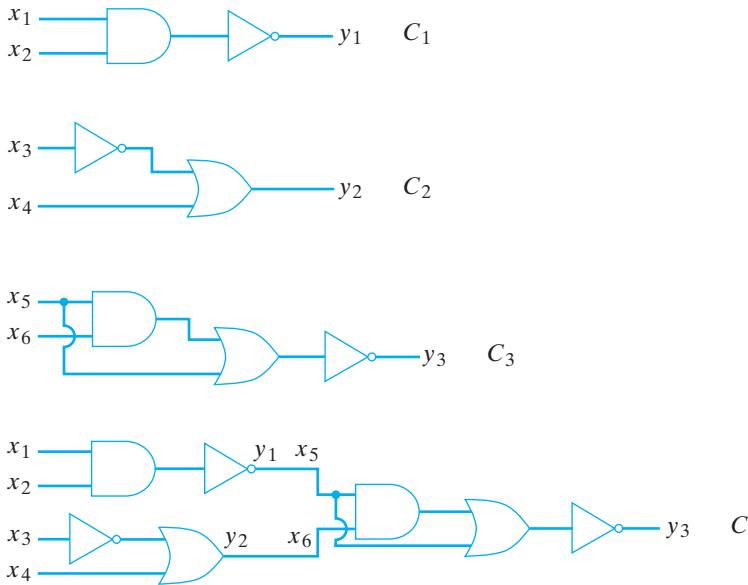


Figure 11.1.6 A circuit that is not a combinational circuit.

**Example 11.1.7**

Individual combinational circuits may be interconnected. The combinational circuits  $C_1$ ,  $C_2$ , and  $C_3$  of Figure 11.1.7 may be combined, as shown, to obtain the combinational circuit  $C$ .



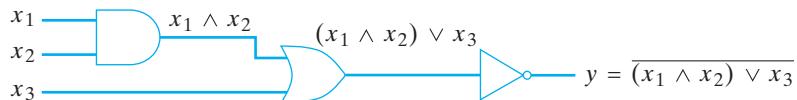
**Figure 11.1.7** Combinatorial circuit  $C$  is obtained by interconnecting the combinational circuits  $C_1$ ,  $C_2$ , and  $C_3$ .

### Example 11.1.8

A combinational circuit with one output, such as that in Figure 11.1.4, can be represented by an expression using the symbols  $\wedge$ ,  $\vee$ , and  $\neg$ . We follow the flow of the circuit symbolically. First,  $x_1$  and  $x_2$  are ANDed (see Figure 11.1.8), which produces output  $x_1 \wedge x_2$ . This output is then ORed with  $x_3$  to produce output  $(x_1 \wedge x_2) \vee x_3$ . This output is then NOTed. Thus the output  $y$  may be

$$y = \overline{(x_1 \wedge x_2) \vee x_3}. \quad (11.1.1)$$

Expressions such as (11.1.1) are called **Boolean expressions**.



**Figure 11.1.8** Representation of a combinational circuit by a Boolean expression.

**Definition 11.1.9** ► Boolean expressions in the symbols  $x_1, \dots, x_n$  are defined recursively as follows.

$$0, 1, x_1, \dots, x_n \quad (11.1.2)$$

are Boolean expression. If  $X_1$  and  $X_2$  are Boolean expressions, then

$$(a) (X_1), \quad (b) \overline{X_1}, \quad (c) X_1 \vee X_2, \quad (d) X_1 \wedge X_2 \quad (11.1.3)$$

are Boolean expressions.

If  $X$  is a Boolean expression in the symbols  $x_1, \dots, x_n$ , we sometimes write

$$X = X(x_1, \dots, x_n).$$

Either symbol  $x$  or  $\bar{x}$  is called a *literal*.

**Example 11.1.10** Use Definition 11.1.9 to show that the right side of (11.1.1) is a Boolean expression in  $x_1$ ,  $x_2$ , and  $x_3$ .

**SOLUTION** By (11.1.2),  $x_1$  and  $x_2$  are Boolean expressions. By (11.1.3d),  $x_1 \wedge x_2$  is a Boolean expression. By (11.1.3a),  $(x_1 \wedge x_2)$  is a Boolean expression. By (11.1.2),  $x_3$  is a Boolean expression. Since  $(x_1 \wedge x_2)$  and  $x_3$  are Boolean expressions, by (11.1.3c), so is  $(x_1 \wedge x_2) \vee x_3$ . Finally, we may apply (11.1.3b) to conclude that  $(x_1 \wedge x_2) \vee x_3$  is a Boolean expression. ▲

If  $X = X(x_1, \dots, x_n)$  is a Boolean expression and  $x_1, \dots, x_n$  are assigned values  $a_1, \dots, a_n$  in  $\{0, 1\}$ , we may use Definitions 11.1.1–11.1.3 to compute a value for  $X$ . We denote this value  $X(a_1, \dots, a_n)$  or  $X(x_i = a_i)$ .

**Example 11.1.11** For  $x_1 = 1$ ,  $x_2 = 0$ , and  $x_3 = 0$ , the Boolean expression  $X(x_1, x_2, x_3) = \overline{(x_1 \wedge x_2) \vee x_3}$  of (11.1.1) becomes

$$\begin{aligned} X(1, 0, 0) &= \overline{(1 \wedge 0) \vee 0} \\ &= \overline{0 \vee 0} \quad \text{since } 1 \wedge 0 = 0 \\ &= \overline{0} \quad \text{since } 0 \vee 0 = 0 \\ &= 1 \quad \text{since } \overline{0} = 1. \end{aligned}$$

We have again computed the fourth row of the table in Example 11.1.5. ▲

In a Boolean expression in which parentheses are not used to specify the order of operations, we assume that  $\wedge$  is evaluated before  $\vee$ .

**Example 11.1.12** For  $x_1 = 0$ ,  $x_2 = 0$ , and  $x_3 = 1$ , the value of the Boolean expression  $x_1 \wedge x_2 \vee x_3$  is

$$x_1 \wedge x_2 \vee x_3 = 0 \wedge 0 \vee 1 = 0 \vee 1 = 1.$$
 ▲

Example 11.1.8 showed how to represent a combinatorial circuit with one output as a Boolean expression. The following example shows how to construct a combinatorial circuit that represents a Boolean expression.

**Example 11.1.13** Find the combinatorial circuit corresponding to the Boolean expression

$$(x_1 \wedge (\bar{x}_2 \vee x_3)) \vee x_2$$

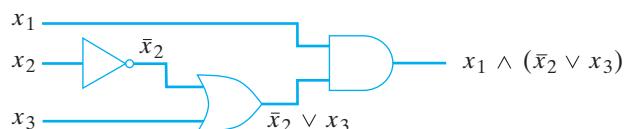
and write the logic table for the circuit obtained.

**SOLUTION** We begin with the expression  $\bar{x}_2 \vee x_3$  in the innermost parentheses. This expression is converted to a combinatorial circuit, as shown in Figure 11.1.9.

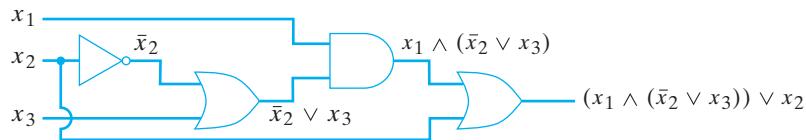
The output of this circuit is ANDed with  $x_1$  to produce the circuit drawn in Figure 11.1.10. Finally, the output of this circuit is ORed with  $x_2$  to give the desired circuit drawn in Figure 11.1.11. The logic table follows.



**Figure 11.1.9** The combinatorial circuit corresponding to the Boolean expression  $\bar{x}_2 \vee x_3$ .



**Figure 11.1.10** The combinatorial circuit corresponding to the Boolean expression  $x_1 \wedge (\bar{x}_2 \vee x_3)$ .



**Figure 11.1.11** The combinational circuit corresponding to the Boolean expression  $(x_1 \wedge (\bar{x}_2 \vee x_3)) \vee x_2$ .

| $x_1$ | $x_2$ | $x_3$ | $(x_1 \wedge (\bar{x}_2 \vee x_3)) \vee x_2$ |
|-------|-------|-------|----------------------------------------------|
| 1     | 1     | 1     | 1                                            |
| 1     | 1     | 0     | 1                                            |
| 1     | 0     | 1     | 1                                            |
| 1     | 0     | 0     | 1                                            |
| 0     | 1     | 1     | 1                                            |
| 0     | 1     | 0     | 1                                            |
| 0     | 0     | 1     | 0                                            |
| 0     | 0     | 0     | 0                                            |



## 11.1 Review Exercises

1. What is a combinational circuit?
2. What is a sequential circuit?
3. What is an AND gate?
4. What is an OR gate?
5. What is a NOT gate?
6. What is an inverter?
7. What is a logic table of a combinational circuit?
8. What is a Boolean expression?
9. What is a literal?

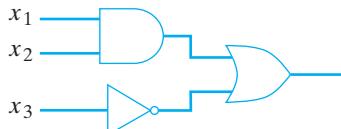
## 11.1 Exercises

In Exercises 1–6, write the Boolean expression that represents the combinational circuit, write the logic table, and write the output of each gate symbolically as in Figure 11.1.8.

1.



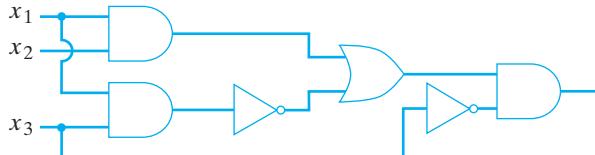
2.



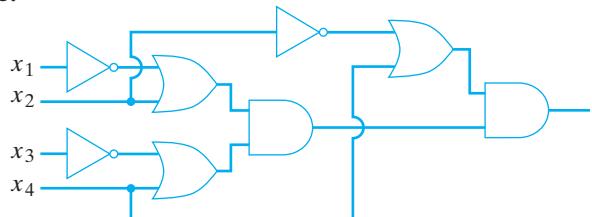
3.



4.



5.



6. The circuit at the bottom of Figure 11.1.7.

Exercises 7–9 refer to the circuit



7. Show that this circuit is not a combinational circuit.
8. Show that if  $x = 0$ , the output  $y$  is uniquely determined.
9. Show that if  $x = 1$ , the output  $y$  is undetermined.

In Exercises 10–14, find the value of the Boolean expressions for

$$x_1 = 1, \quad x_2 = 1, \quad x_3 = 0, \quad x_4 = 1.$$

10.  $\overline{x_1 \wedge x_2}$

11.  $(x_1 \wedge \bar{x}_2) \vee (x_1 \vee \bar{x}_3)$

12.  $x_1 \vee (\bar{x}_2 \wedge x_3)$   
 13.  $(x_1 \wedge (x_2 \vee (x_1 \wedge \bar{x}_2))) \vee ((x_1 \wedge \bar{x}_2) \vee (\overline{x_1 \wedge \bar{x}_3}))$   
 14.  $((x_1 \wedge x_2) \vee (x_3 \wedge \bar{x}_4)) \vee (\overline{(x_1 \vee x_3)} \wedge (\bar{x}_2 \vee x_3)) \vee (x_1 \wedge \bar{x}_3)$   
 15. Using Definition 11.1.9, show that each expression in Exercises 10–14 is a Boolean expression.

In Exercises 16–20, tell whether the given expression is a Boolean expression. If it is a Boolean expression, use Definition 11.1.9 to show that it is.

16.  $x_1 \wedge (x_2 \vee x_3)$   
 17.  $(x_1)$   
 18.  $x_1 \wedge \bar{x}_2 \vee x_3$   
 19.  $((x_1 \wedge x_2) \vee \bar{x}_3)$   
 20.  $((x_1))$

21. Find the combinatorial circuit corresponding to each Boolean expression in Exercises 10–14 and write the logic table.

A switching circuit is an electrical network consisting of switches each of which is open or closed. An example is given in Figure 11.1.12. If switch  $X$  is open (closed), we write  $X = 0$  ( $X = 1$ ). Switches labeled with the same letter, such as  $B$  in Figure 11.1.12, are either all open or all closed. Switch  $X$ , such as  $A$  in Figure 11.1.12, is open if and only if switch  $\bar{X}$ , such as  $\bar{A}$ , is closed. If current can flow between the extreme left and right ends of the circuit, we say that the output of the circuit is 1; otherwise, we say that the output of the circuit is 0. A switching table gives the output of the circuit for all values of the switches. The switching table for Figure 11.1.12 is as follows:

| A | B | C | Circuit Output |
|---|---|---|----------------|
| 1 | 1 | 1 | 1              |
| 1 | 1 | 0 | 1              |
| 1 | 0 | 1 | 0              |
| 1 | 0 | 0 | 0              |
| 0 | 1 | 1 | 1              |
| 0 | 1 | 0 | 1              |
| 0 | 0 | 1 | 1              |
| 0 | 0 | 0 | 1              |

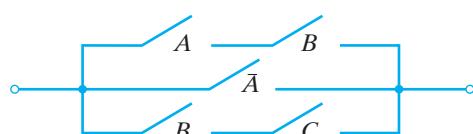


Figure 11.1.12 A switching circuit.

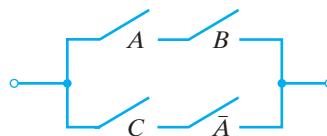
22. Draw a circuit with two switches  $A$  and  $B$  having the property that the circuit output is 1 precisely when both  $A$  and  $B$  are closed. This configuration is labeled  $A \wedge B$  and is called a **series circuit**.

23. Draw a circuit with two switches  $A$  and  $B$  having the property that the circuit output is 1 precisely when either  $A$  or  $B$  is closed. This configuration is labeled  $A \vee B$  and is called a **parallel circuit**.
24. Show that the circuit of Figure 11.1.12 can be represented symbolically as

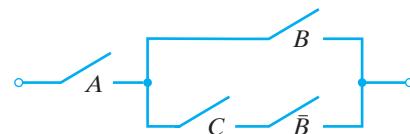
$$(A \wedge B) \vee \bar{A} \vee (B \wedge C).$$

Represent each circuit in Exercises 25–29 symbolically and give its switching table.

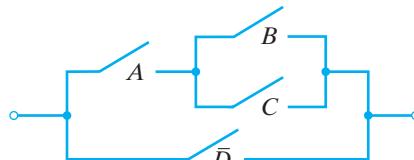
25.



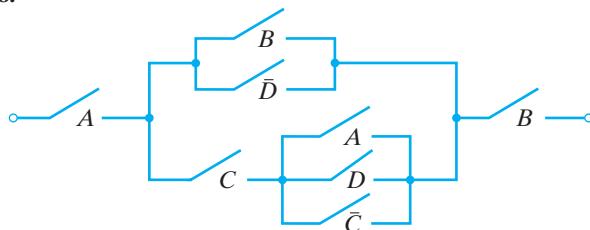
26.



27.



28.



Represent the expressions in Exercises 29–33 as switching circuits and write the switching tables.

29.  $(A \vee \bar{B}) \wedge A$   
 30.  $A \vee (\bar{B} \wedge C)$   
 31.  $(\bar{A} \wedge B) \vee (C \wedge A)$   
 32.  $(A \wedge ((B \wedge \bar{C}) \vee (\bar{B} \wedge C))) \vee (\bar{A} \wedge B \wedge C)$   
 33.  $A \wedge ((B \wedge C \wedge \bar{D}) \vee (\bar{B} \wedge C \vee D) \vee (\bar{B} \wedge \bar{C} \wedge D)) \wedge (B \vee \bar{D})$

## 11.2 Properties of Combinatorial Circuits

In the preceding section we defined two binary operators  $\wedge$  and  $\vee$  on  $Z_2 = \{0, 1\}$  and a unary operator  $\bar{\phantom{x}}$  on  $Z_2$ . (Throughout the remainder of this chapter we let  $Z_2$  denote the set  $\{0, 1\}$ .) We saw that these operators could be implemented in circuits as gates. In this section we discuss some properties of the system consisting of  $Z_2$  and the operators  $\wedge$ ,  $\vee$ , and  $\bar{\phantom{x}}$ .

### Theorem 11.2.1

If  $\wedge$ ,  $\vee$ , and  $\bar{\phantom{x}}$  are as in Definitions 11.1.1–11.1.3, then the following properties hold.

(a) Associative laws:

$$\begin{aligned} (a \vee b) \vee c &= a \vee (b \vee c) \\ (a \wedge b) \wedge c &= a \wedge (b \wedge c) \end{aligned} \quad \text{for all } a, b, c \in Z_2.$$

(b) Commutative laws:

$$a \vee b = b \vee a, \quad a \wedge b = b \wedge a \quad \text{for all } a, b \in Z_2.$$

(c) Distributive laws:

$$\begin{aligned} a \wedge (b \vee c) &= (a \wedge b) \vee (a \wedge c) \\ a \vee (b \wedge c) &= (a \vee b) \wedge (a \vee c) \end{aligned} \quad \text{for all } a, b, c \in Z_2.$$

(d) Identity laws:

$$a \vee 0 = a, \quad a \wedge 1 = a \quad \text{for all } a \in Z_2.$$

(e) Complement laws:

$$a \vee \bar{a} = 1, \quad a \wedge \bar{a} = 0 \quad \text{for all } a \in Z_2.$$

**Proof** The proofs are straightforward verifications. We shall prove the first distributive law only and leave the other equations as exercises (see Exercises 16 and 17).

We must show that

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c) \quad \text{for all } a, b, c \in Z_2. \quad (11.2.1)$$

We simply evaluate both sides of (11.2.1) for all possible values of  $a$ ,  $b$ , and  $c$  in  $Z_2$  and verify that in each case we obtain the same result. The table gives the details.

| $a$ | $b$ | $c$ | $a \wedge (b \vee c)$ | $(a \wedge b) \vee (a \wedge c)$ |
|-----|-----|-----|-----------------------|----------------------------------|
| 1   | 1   | 1   | 1                     | 1                                |
| 1   | 1   | 0   | 1                     | 1                                |
| 1   | 0   | 1   | 1                     | 1                                |
| 1   | 0   | 0   | 0                     | 0                                |
| 0   | 1   | 1   | 0                     | 0                                |
| 0   | 1   | 0   | 0                     | 0                                |
| 0   | 0   | 1   | 0                     | 0                                |
| 0   | 0   | 0   | 0                     | 0                                |

**Example 11.2.2** By using Theorem 11.2.1, show that the combinatorial circuits of Figure 11.2.1 have identical outputs for given identical inputs.



**Figure 11.2.1** The combinatorial circuits (a) and (b) have identical outputs for given identical inputs and are said to be equivalent.

**SOLUTION** The Boolean expressions representing the circuits are, respectively,

$$x_1 \vee (x_2 \wedge x_3), \quad (x_1 \vee x_2) \wedge (x_1 \vee x_3).$$

By Theorem 11.2.1(c),

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c) \quad \text{for all } a, b, c \in Z_2. \quad (11.2.2)$$

But (11.2.2) says that the combinatorial circuits of Figure 11.2.1 have identical outputs for given identical inputs. ▲

Arbitrary Boolean expressions are defined to be equal if they have the same values for all possible assignments of bits to the literals.

**Definition 11.2.3** ▶ Let

$$X_1 = X_1(x_1, \dots, x_n) \quad \text{and} \quad X_2 = X_2(x_1, \dots, x_n)$$

be Boolean expressions. We define  $X_1$  to be *equal* to  $X_2$  and write  $X_1 = X_2$  if

$$X_1(a_1, \dots, a_n) = X_2(a_1, \dots, a_n) \quad \text{for all } a_i \in Z_2. \quad \blacktriangleleft$$

**Example 11.2.4** Show that

$$(\bar{x} \vee \bar{y}) = \bar{x} \wedge \bar{y}. \quad (11.2.3)$$

**SOLUTION** According to Definition 11.2.3, equation (11.2.3) holds if the equation is true for all choices of  $x$  and  $y$  in  $Z_2$ . Thus we may simply construct a table listing all possibilities to verify (11.2.3).

| $x$ | $y$ | $(\bar{x} \vee \bar{y})$ | $\bar{x} \wedge \bar{y}$ |
|-----|-----|--------------------------|--------------------------|
| 1   | 1   | 0                        | 0                        |
| 1   | 0   | 0                        | 0                        |
| 0   | 1   | 0                        | 0                        |
| 0   | 0   | 1                        | 1                        |

If we define a relation  $R$  on a set of Boolean expressions by the rule  $X_1 R X_2$  if  $X_1 = X_2$ ,  $R$  is an equivalence relation. Each equivalence class consists of a set of Boolean expressions any one of which is equal to any other.

Because of the associative laws, Theorem 11.2.1(a), we can unambiguously write

$$a_1 \vee a_2 \vee \cdots \vee a_n \quad (11.2.4)$$

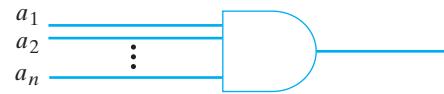
or

$$a_1 \wedge a_2 \wedge \cdots \wedge a_n \quad (11.2.5)$$

for  $a_i \in Z_2$ . The combinatorial circuit corresponding to (11.2.4) is drawn as in Figure 11.2.2, and the combinatorial circuit corresponding to (11.2.5) is drawn as in Figure 11.2.3.



**Figure 11.2.2** An  $n$ -input OR gate.



**Figure 11.2.3** An  $n$ -input AND gate.

The properties listed in Theorem 11.2.1 hold for a variety of systems. Any system satisfying these properties is called a **Boolean algebra**. Abstract Boolean algebras are examined in Section 11.3.

Having defined equality of Boolean expressions, we define equivalence of combinatorial circuits.

**Definition 11.2.5** ► We say that two combinatorial circuits, each having inputs  $x_1, \dots, x_n$  and a single output, are *equivalent* if, whenever the circuits receive the same inputs, they produce the same outputs. ◀

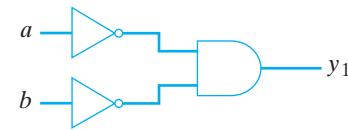
### Example 11.2.6

The combinatorial circuits of Figures 11.2.4 and 11.2.5 are equivalent since, as shown, they have identical logic tables.



| $a$ | $b$ | $y_1$ |
|-----|-----|-------|
| 1   | 1   | 0     |
| 1   | 0   | 0     |
| 0   | 1   | 0     |
| 0   | 0   | 1     |

**Figure 11.2.4** A combinatorial circuit and its logic table.



| $a$ | $b$ | $y_1$ |
|-----|-----|-------|
| 1   | 1   | 0     |
| 1   | 0   | 0     |
| 0   | 1   | 0     |
| 0   | 0   | 1     |

**Figure 11.2.5** A combinatorial circuit and its logic table, which is identical to the logic table of Figure 11.2.4. The circuits of Figures 11.2.4 and 11.2.5 are said to be equivalent because they have identical logic tables. ◀

If we define a relation  $R$  on a set of combinatorial circuits by the rule  $C_1 R C_2$  if  $C_1$  and  $C_2$  are equivalent (in the sense of Definition 11.2.5),  $R$  is an equivalence relation. Each equivalence class consists of a set of mutually equivalent combinatorial circuits.

Example 11.2.6 shows that equivalent circuits may not have the same number of gates. In general, it is desirable to use as few gates as possible to minimize the cost of the components.

It follows immediately from the definitions that combinatorial circuits are equivalent if and only if the Boolean expressions that represent them are equal.

### Theorem 11.2.7

Let  $C_1$  and  $C_2$  be combinatorial circuits represented, respectively, by the Boolean expressions  $X_1 = X_1(x_1, \dots, x_n)$  and  $X_2 = X_2(x_1, \dots, x_n)$ . Then  $C_1$  and  $C_2$  are equivalent if and only if  $X_1 = X_2$ .

**Proof** The value  $X_1(a_1, \dots, a_n)$  [respectively,  $X_2(a_1, \dots, a_n)$ ] for  $a_i \in Z_2$  is the output for circuit  $C_1$  (respectively,  $C_2$ ) for inputs  $a_1, \dots, a_n$ .

According to Definition 11.2.5, circuits  $C_1$  and  $C_2$  are equivalent if and only if they have the same outputs  $X_1(a_1, \dots, a_n)$  and  $X_2(a_1, \dots, a_n)$  for all possible inputs  $a_1, \dots, a_n$ . Thus circuits  $C_1$  and  $C_2$  are equivalent if and only if

$$X_1(a_1, \dots, a_n) = X_2(a_1, \dots, a_n) \quad \text{for all values } a_i \in Z_2. \quad (11.2.6)$$

But by Definition 11.2.3, equation (11.2.6) holds if and only if  $X_1 = X_2$ . ◀

### Example 11.2.8

In Example 11.2.4 we showed that

$$\overline{(x \vee y)} = \bar{x} \wedge \bar{y}.$$

By Theorem 11.2.7, the combinatorial circuits (Figures 11.2.4 and 11.2.5) corresponding to these expressions are equivalent. ◀

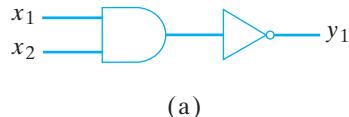
## 11.2 Review Exercises

1. State the associative laws for  $\wedge$  and  $\vee$ .
2. State the commutative laws for  $\wedge$  and  $\vee$ .
3. State the distributive laws for  $\wedge$  and  $\vee$ .
4. State the identity laws for  $\wedge$  and  $\vee$ .
5. State the complement laws for  $\wedge$ ,  $\vee$ , and  $\neg$ .
6. When are two Boolean expressions equal?
7. What are equivalent combinatorial expressions?
8. What is the relation between combinatorial expressions and the Boolean expressions that represent them?

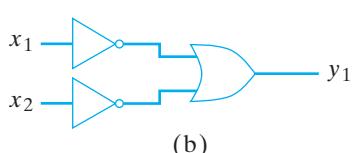
## 11.2 Exercises

Show that the combinatorial circuits of Exercises 1–5 are equivalent.

1.

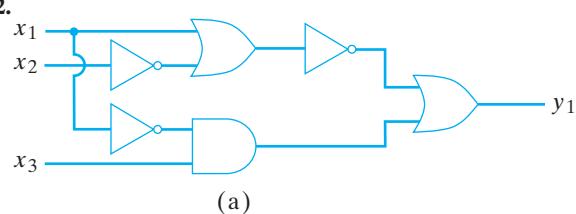


(a)

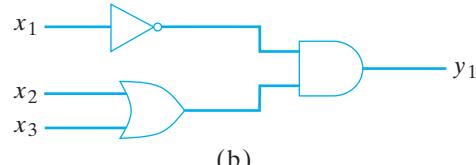


(b)

2.

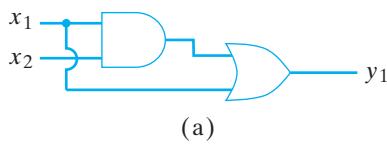


(a)

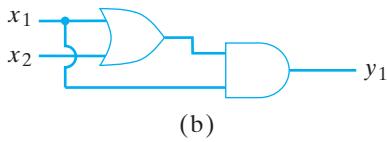


(b)

3.

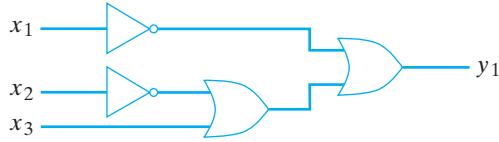


(a)

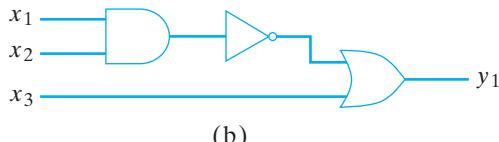


(b)

4.

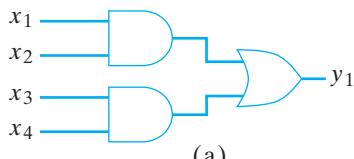


(a)

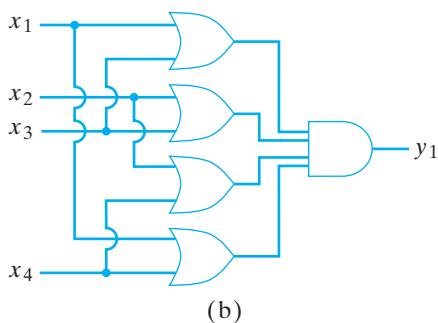


(b)

5.



(a)



(b)

Verify the equations in Exercises 6–10.

6.  $x_1 \vee x_1 = x_1$

7.  $x_1 \wedge \bar{x}_2 = (\bar{x}_1 \vee x_2)$

8.  $x_1 \vee (x_1 \wedge x_2) = x_1$

9.  $x_1 \wedge (\bar{x}_2 \wedge \bar{x}_3) = (x_1 \wedge \bar{x}_2) \vee (x_1 \wedge \bar{x}_3)$

10.  $(x_1 \vee x_2) \wedge (x_3 \vee x_4) = (x_3 \wedge x_1) \vee (x_3 \wedge x_2) \vee (x_4 \wedge x_1) \vee (x_4 \wedge x_2)$

Prove or disprove the equations in Exercises 11–15.

11.  $\bar{\bar{x}} = x$

12.  $\bar{x}_1 \wedge \bar{x}_2 = x_1 \vee x_2$

13.  $\bar{x}_1 \wedge ((x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3)) = x_2 \wedge x_3$

14.  $((\bar{x}_1 \wedge x_2) \vee (x_1 \wedge \bar{x}_3)) = (x_1 \vee \bar{x}_2) \wedge (x_1 \vee \bar{x}_3)$

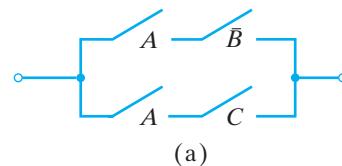
15.  $(x_1 \vee x_2) \wedge (\bar{x}_3 \vee x_4) \wedge (x_3 \wedge \bar{x}_2) = 0$

16. Prove the second statement of Theorem 11.2.1(c).

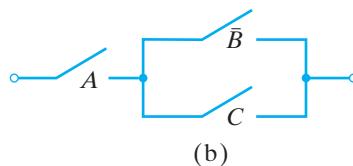
17. Prove Theorem 11.2.1, parts (a), (b), (d), and (e).

We say that two switching circuits are equivalent if the Boolean expressions that represent them are equal.

18. Show that the switching circuits are equivalent.



(a)

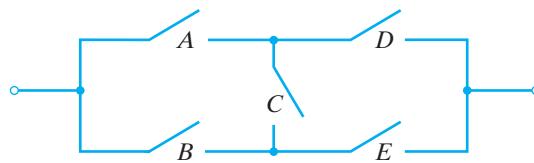


(b)

19. For each switching circuit in Exercises 25–28, Section 11.1, find an equivalent switching circuit using parallel and series circuits having as few switches as you can.

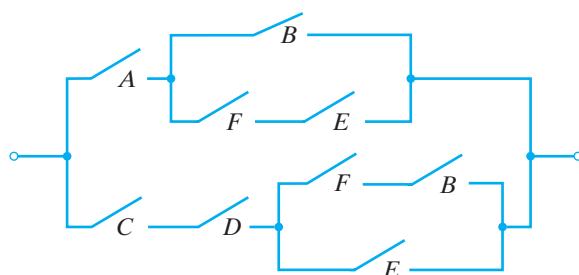
20. For each Boolean expression in Exercises 29–33, Section 11.1, find a switching circuit using parallel and series circuits having as few switches as you can.

A bridge circuit is a switching circuit, such as the one shown here, that uses nonparallel and nonseries circuits.

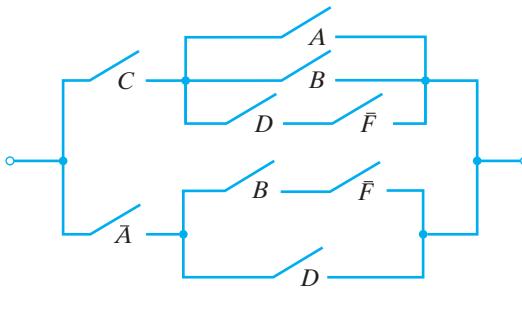


For each switching circuit, find an equivalent switching circuit using bridge circuits having as few switches as you can.

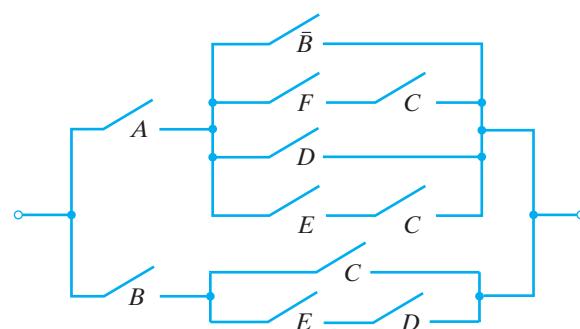
21.



22.



★23.



24. For each Boolean expression in Exercises 29–33, Section 11.1, find a switching circuit using bridge circuits having as few switches as you can.

## 11.3 Boolean Algebras

### Go Online

For more on Boolean algebras, see  
[goo.gl/5HWh0y](http://goo.gl/5HWh0y)

In this section we consider general systems that have properties like those given in Theorem 11.2.1. We will see that apparently diverse systems obey these same laws (for example, see Example 11.3.2, Exercises 2 and 22, and Exercise 8 following the Problem-Solving Corner). We call such systems Boolean algebras.

**Definition 11.3.1** ► A Boolean algebra  $B$  consists of a set  $S$  containing distinct elements 0 and 1, binary operators  $+$  and  $\cdot$  on  $S$ , and a unary operator  $'$  on  $S$  satisfying the following laws.

(a) Associative laws:

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

$$(x \cdot y) \cdot z = x \cdot (y \cdot z) \quad \text{for all } x, y, z \in S.$$

(b) Commutative laws:

$$x + y = y + x, \quad x \cdot y = y \cdot x \quad \text{for all } x, y \in S.$$

(c) Distributive laws:

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

$$x + (y \cdot z) = (x + y) \cdot (x + z) \quad \text{for all } x, y, z \in S.$$

(d) Identity laws:

$$x + 0 = x, \quad x \cdot 1 = x \quad \text{for all } x \in S.$$

(e) Complement laws:

$$x + x' = 1, \quad x \cdot x' = 0 \quad \text{for all } x \in S.$$

If  $B$  is a Boolean algebra, we write  $B = (S, +, \cdot, ', 0, 1)$ .

The associative laws can be omitted from Definition 11.3.1 since they follow from the other laws (see Exercise 24).

**Example 11.3.2**

By Theorem 11.2.1,  $(Z_2, \vee, \wedge, \neg, 0, 1)$  is a Boolean algebra. (We are letting  $Z_2$  denote the set  $\{0, 1\}$ .) The operators  $+, \cdot, '$  in Definition 11.3.1 are  $\vee, \wedge, \neg$ , respectively. ◀

As is the standard custom, we will usually abbreviate  $a \cdot b$  as  $ab$ . We also assume that  $\cdot$  is evaluated before  $+$ . This allows us to eliminate some parentheses. For example, we can write  $(xy) + z$  more simply as  $xy + z$ .

Several comments are in order concerning Definition 11.3.1. In the first place, 0 and 1 are merely symbolic names and, in general, have nothing to do with the numbers 0 and 1. This same comment applies to  $+$  and  $\cdot$ , which merely denote binary operators and, in general, have nothing to do with ordinary addition and multiplication.

**Example 11.3.3**

Let  $U$  be a universal set and let  $S = \mathcal{P}(U)$ , the power set of  $U$ . If we define the following operations

$$X + Y = X \cup Y, \quad X \cdot Y = X \cap Y, \quad X' = \bar{X}$$

on  $S$ , then  $(S, \cup, \cap, \neg, \emptyset, U)$  is a Boolean algebra. The empty set  $\emptyset$  plays the role of 0 and the universal set  $U$  plays the role of 1. If we let  $X, Y$ , and  $Z$  be subsets of  $S$ , properties (a)–(e) of Definition 11.3.1 become the following properties of sets (see Theorem 1.1.22):

- (a')  $(X \cup Y) \cup Z = X \cup (Y \cup Z)$   
 $(X \cap Y) \cap Z = X \cap (Y \cap Z)$  for all  $X, Y, Z \in \mathcal{P}(U)$ .
- (b')  $X \cup Y = Y \cup X, \quad X \cap Y = Y \cap X$  for all  $X, Y \in \mathcal{P}(U)$ .
- (c')  $X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$   
 $X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$  for all  $X, Y, Z \in \mathcal{P}(U)$ .
- (d')  $X \cup \emptyset = X, \quad X \cap U = X$  for every  $X \in \mathcal{P}(U)$ .
- (e')  $X \cup \bar{X} = U, \quad X \cap \bar{X} = \emptyset$  for every  $X \in \mathcal{P}(U)$ . ◀

At this point we will deduce several other properties of Boolean algebras. We begin by showing that the element  $x'$  in Definition 11.3.1(e) is unique.

**Theorem 11.3.4**

In a Boolean algebra, the element  $x'$  of Definition 11.3.1(e) is unique. Specifically, if  $x + y = 1$  and  $xy = 0$ , then  $y = x'$ .

**Proof**

$$\begin{aligned}
 y &= y1 && \text{Definition 11.3.1(d)} \\
 &= y(x + x') && \text{Definition 11.3.1(e)} \\
 &= yx + yx' && \text{Definition 11.3.1(c)} \\
 &= xy + yx' && \text{Definition 11.3.1(b)} \\
 &= 0 + yx' && \text{Given} \\
 &= yx' && \text{Definition 11.3.1(e)} \\
 &= xx' + yx' && \text{Definition 11.3.1(b)} \\
 &= x'x + x'y && \text{Definition 11.3.1(b)} \\
 &= x'(x + y) && \text{Definition 11.3.1(c)} \\
 &= x'1 && \text{Given} \\
 &= x' && \text{Definition 11.3.1(d)} \quad \blacktriangleleft
 \end{aligned}$$

**Definition 11.3.5** ► In a Boolean algebra, we call the element  $x'$  the *complement* of  $x$ . ◀

We can now derive several additional properties of Boolean algebras.

**Theorem 11.3.6** Let  $B = (S, +, \cdot, ', 0, 1)$  be a Boolean algebra. The following properties hold.

(a) Idempotent laws:

$$x + x = x, \quad xx = x \quad \text{for all } x \in S.$$

(b) Bound laws:

$$x + 1 = 1, \quad x0 = 0 \quad \text{for all } x \in S.$$

(c) Absorption laws:

$$x + xy = x, \quad x(x + y) = x \quad \text{for all } x, y \in S.$$

(d) Involution law:

$$(x')' = x \quad \text{for all } x \in S.$$

(e) 0 and 1 laws:

$$0' = 1, \quad 1' = 0.$$

(f) De Morgan's laws for Boolean algebras:

$$(x + y)' = x'y', \quad (xy)' = x' + y' \quad \text{for all } x, y \in S.$$

**Proof** We will prove (b) and the first statement of parts (a), (c), and (f) and leave the others as exercises (see Exercises 18–20).

|     |                     |                      |
|-----|---------------------|----------------------|
| (a) | $x = x + 0$         | Definition 11.3.1(d) |
|     | $= x + (xx')$       | Definition 11.3.1(e) |
|     | $= (x + x)(x + x')$ | Definition 11.3.1(c) |
|     | $= (x + x)1$        | Definition 11.3.1(e) |
|     | $= x + x$           | Definition 11.3.1(d) |
| (b) | $x + 1 = (x + 1)1$  | Definition 11.3.1(d) |
|     | $= (x + 1)(x + x')$ | Definition 11.3.1(e) |
|     | $= x + 1x'$         | Definition 11.3.1(c) |
|     | $= x + x'1$         | Definition 11.3.1(b) |
|     | $= x + x'$          | Definition 11.3.1(d) |
|     | $= 1$               | Definition 11.3.1(e) |
|     | $x0 = x0 + 0$       | Definition 11.3.1(d) |
|     | $= x0 + xx'$        | Definition 11.3.1(e) |
|     | $= x(0 + x')$       | Definition 11.3.1(c) |
|     | $= x(x' + 0)$       | Definition 11.3.1(b) |
|     | $= xx'$             | Definition 11.3.1(d) |
|     | $= 0$               | Definition 11.3.1(e) |
| (c) | $x + xy = x1 + xy$  | Definition 11.3.1(d) |
|     | $= x(1 + y)$        | Definition 11.3.1(c) |
|     | $= x(y + 1)$        | Definition 11.3.1(b) |
|     | $= x1$              | Part (b)             |
|     | $= x$               | Definition 11.3.1(d) |

(f) If we show that

$$(x + y)(x'y') = 0 \quad (11.3.1)$$

and

$$(x + y) + x'y' = 1, \quad (11.3.2)$$

it will follow from Theorem 11.3.4 that  $x'y' = (x + y)'$ . Now

$$\begin{aligned} (x + y)(x'y') &= (x'y')(x + y) && \text{Definition 11.3.1(b)} \\ &= (x'y')x + (x'y')y && \text{Definition 11.3.1(c)} \\ &= x(x'y') + (x'y')y && \text{Definition 11.3.1(b)} \\ &= (xx')y' + x'(y'y) && \text{Definition 11.3.1(a)} \\ &= (xx')y' + x'(yy') && \text{Definition 11.3.1(b)} \\ &= 0y' + x'0 && \text{Definition 11.3.1(e)} \\ &= y'0 + x'0 && \text{Definition 11.3.1(b)} \\ &= 0 + 0 && \text{Part (b)} \\ &= 0 && \text{Definition 11.3.1(d)} \end{aligned}$$

Therefore, (11.3.1) holds.

Next we verify (11.3.2).

$$\begin{aligned} (x + y) + x'y' &= ((x + y) + x')((x + y) + y') && \text{Definition 11.3.1(c)} \\ &= ((y + x) + x')((x + y) + y') && \text{Definition 11.3.1(b)} \\ &= (y + (x + x'))(x + (y + y')) && \text{Definition 11.3.1(a)} \\ &= (y + 1)(x + 1) && \text{Definition 11.3.1(e)} \\ &= 1 \cdot 1 && \text{Part (b)} \\ &= 1 && \text{Definition 11.3.1(d)} \end{aligned}$$

By Theorem 11.3.4,  $x'y' = (x + y)'$ . ◀

### Example 11.3.7

As explained in Example 11.3.3, if  $U$  is a set,  $\mathcal{P}(U)$  can be considered a Boolean algebra. Therefore, De Morgan's laws, which for sets may be stated

$$(\overline{X \cup Y}) = \overline{X} \cap \overline{Y}, \quad (\overline{X \cap Y}) = \overline{X} \cup \overline{Y} \quad \text{for all } X, Y \in \mathcal{P}(U),$$

hold. These equations may be verified directly (see Theorem 1.1.22), but Theorem 11.3.6 shows that they are a consequence of other laws. ◀

The reader has surely noticed that equations involving elements of a Boolean algebra come in pairs. For example, the identity laws [Definition 11.3.1(d)] are

$$x + 0 = x, \quad x1 = x.$$

Such pairs are said to be **dual**.

**Definition 11.3.8** ► The *dual* of a statement involving Boolean expressions is obtained by replacing 0 by 1, 1 by 0, + by  $\cdot$ , and  $\cdot$  by  $+$ . ◀

### Example 11.3.9

The dual of  $(x + y)' = x'y'$  is  $(xy)' = x' + y'$ . ◀

Each condition in the definition of a Boolean algebra (Definition 11.3.1) includes its dual. Therefore, we have the following result.

### Theorem 11.3.10

The dual of a theorem about Boolean algebras is also a theorem.

**Proof** Suppose that  $T$  is a theorem about Boolean algebras. Then there is a proof  $P$  of  $T$  involving only the definitions of a Boolean algebra (Definition 11.3.1). Let  $P'$  be

the sequence of statements obtained by replacing every statement in  $P$  by its dual. Then  $P'$  is a proof of the dual of  $T$ . 

**Example 11.3.11** The dual of

$$x + x = x \quad (11.3.3)$$

is

$$xx = x. \quad (11.3.4)$$

We proved (11.3.3) earlier [see the proof of Theorem 11.3.6(a)]. If we write the dual of each statement in the proof of (11.3.3), we obtain the following proof of (11.3.4):

$$\begin{aligned} x &= x1 \\ &= x(x + x') \\ &= xx + xx' \\ &= xx + 0 \\ &= xx. \end{aligned}$$


**Example 11.3.12** The proofs given in Theorem 11.3.6 of the two statements of part (b) are dual to each other. 

## 11.3 Review Exercises

1. Define Boolean algebra.
2. What are the idempotent laws for Boolean algebras?
3. What are the bound laws for Boolean algebras?
4. What are the absorption laws for Boolean algebras?
5. What is the involution law for Boolean algebras?
6. What are the 0/1 laws for Boolean algebras?
7. What are De Morgan's laws for Boolean algebras?
8. How is the dual of a Boolean expression obtained?
9. What can we say about the dual of a theorem about Boolean algebras?

## 11.3 Exercises

1. Verify properties (a')–(e') of Example 11.3.3.
2. Let  $S = \{1, 2, 3, 6\}$ . Define
$$x + y = \text{lcm}(x, y), \quad x \cdot y = \gcd(x, y), \quad x' = \frac{6}{x}$$
for  $x, y \in S$  ( $\text{lcm}$  and  $\gcd$  denote, respectively, the least common multiple and the greatest common divisor). Show that  $(S, +, \cdot, ', 1, 6)$  is a Boolean algebra.
3.  $S = \{1, 2, 4, 8\}$ . Define  $+$  and  $\cdot$  as in Exercise 2 and define  $x' = 8/x$ . Show that  $(S, +, \cdot, ', 1, 8)$  is not a Boolean algebra.

Let  $S_n = \{1, 2, \dots, n\}$ . Define

$$x + y = \max\{x, y\}, \quad x \cdot y = \min\{x, y\}.$$

4. Show that parts (a)–(c) of Definition 11.3.1 hold for  $S_n$ .
5. Show that it is possible to define 0, 1 and  $'$  so that  $(S_n, +, \cdot, ', 0, 1)$  is a Boolean algebra if and only if  $n = 2$ .
6. Rewrite the conditions of Theorem 11.3.6 for sets as in Example 11.3.3.
7. Interpret Theorem 11.3.4 for sets as in Example 11.3.3.

*Write the dual of each statement in Exercises 8–14.*

8.  $(x + y)(x + 1) = x + xy + y$
9.  $(x' + y')' = xy$
10. If  $x + y = x + z$  and  $x' + y = x' + z$ , then  $y = z$ .
11.  $xy' = 0$  if and only if  $xy = x$ .
12.  $x = 0$  if and only if  $y = xy' + x'y$  for all  $y$ .
13. If  $x + y = 0$ , then  $x = 0 = y$ .
14.  $x + x(y + 1) = x$
15. Prove the statements of Exercises 8–14.
16. Prove the duals of the statements of Exercises 8–14.
17. Write the dual of Theorem 11.3.4. How does the dual relate to Theorem 11.3.4 itself?
18. Prove the second statements of parts (a), (c), and (f) of Theorem 11.3.6.
19. Prove the second statements of parts (a), (c), and (f) of Theorem 11.3.6 by dualizing the proofs of the first statements given in the text.

20. Prove Theorem 11.3.6, parts (d) and (e).
- \*21. Deduce part (a) of Definition 11.3.1 from parts (b)–(e) of Definition 11.3.1.
22. Let  $U$  be the set of positive integers. Let  $S$  be the collection of subsets  $X$  of  $U$  with either  $X$  or  $\bar{X}$  finite. Show that  $(S, \cup, \cap, \neg, \emptyset, U)$  is a Boolean algebra.

- \*23. Let  $n$  be a positive integer. Let  $S$  be the set of all divisors of  $n$ , including 1 and  $n$ . Define  $+$  and  $\cdot$  as in Exercise 2 and define  $x' = n/x$ . What conditions must  $n$  satisfy so that  $(S, +, \cdot, ', 1, n)$  is a Boolean algebra?
- \*24. Show that the associative laws follow from the other laws of Definition 11.3.1.

## Problem-Solving Corner

### Problem

Let  $(S, +, \cdot, ', 0, 1)$  be a Boolean algebra and let  $A$  be a subset of  $S$ . Show that  $(A, +, \cdot, ', 0, 1)$  is a Boolean algebra if and only if  $1 \in A$  and  $xy' \in A$  for all  $x, y \in A$ .

### Attacking the Problem

Since the given statement is an “if and only if” statement, there are two statements to be proved:

If  $(A, +, \cdot, ', 0, 1)$  is a Boolean algebra,

then  $1 \in A$  and  $xy' \in A$  for all  $x, y \in A$ . (1)

If  $1 \in A$  and  $xy' \in A$  for all  $x, y \in A$ , then

$(A, +, \cdot, ', 0, 1)$  is a Boolean algebra. (2)

To prove (1), we can use the laws as specified by the definition of “Boolean algebra” (Definition 11.3.1) and the laws derived in Theorem 11.3.6 that elements of a Boolean algebra must obey. To prove that  $(A, +, \cdot, ', 0, 1)$  is a Boolean algebra, we will verify that the laws specified by Definition 11.3.1 are satisfied. Before reading on, you should review Definition 11.3.1 and Theorem 11.3.6.

### Finding a Solution

First let’s try to prove (1). We assume that  $(A, +, \cdot, ', 0, 1)$  is a Boolean algebra and prove that

■  $1 \in A$

and

■  $xy' \in A$  for all  $x, y \in A$ .

Definition 11.3.1 says that a Boolean algebra contains 1. Since  $(A, +, \cdot, ', 0, 1)$  is a Boolean algebra,  $1 \in A$ .

Now suppose that  $x, y \in A$ . Definition 11.3.1 says that  $'$  is a unary operator on  $A$ . This means that  $y' \in A$ . Definition 11.3.1 also says that  $\cdot$  is a binary operator on  $A$ . This means that  $xy' \in A$ . This completes the proof of (1).

Now let’s try to prove (2). This time we assume that  $1 \in A$  and  $xy' \in A$  for all  $x, y \in A$  and try to prove

## Boolean Algebras

that  $(A, +, \cdot, ', 0, 1)$  is a Boolean algebra. According to Definition 11.3.1, we must prove that

$A$  contains distinct elements 0 and 1. (3)

$+$  and  $\cdot$  are binary operators on  $A$ . (4)

$'$  is a unary operator on  $A$ . (5)

The associative laws hold. (6)

The commutative laws hold. (7)

The distributive laws hold. (8)

The identity laws hold. (9)

The complement laws hold. (10)

$A$  contains 1 by assumption. To prove (3), we must show that  $0 \in A$ . We have only two assumptions about  $A$ :  $1 \in A$  and if  $x, y \in A$ , then  $xy' \in A$ . All we can do at this point is combine these assumptions; that is, take  $x = y = 1$  and examine the conclusion:  $1' \in A$ . Now Theorem 11.3.6(e) [applied to the Boolean algebra  $(S, +, \cdot, ', 0, 1)$ ] says that  $1' = 0$ . Substituting for  $1'$ , we know now that  $10 \in A$ . But Theorem 11.3.6(b) says that for any  $x$ ,  $x0 = 0$ . Thus  $10 = 0$  is in  $A$ . Success!  $A$  contains 1 and 0. 0 and 1 are distinct because they are elements of the Boolean algebra  $(S, +, \cdot, ', 0, 1)$ . Therefore, (3) is proved.

To prove (4), we must show that  $+$  and  $\cdot$  are binary operators on  $A$ ; that is, if  $x, y \in A$ , then  $x + y$  and  $xy$  are in  $A$ . Consider proving that  $\cdot$  is a binary operator on  $A$ . What we know is that if  $x, y \in A$ , then  $xy' \in A$ , which is close to what we want to prove. If we could somehow replace  $y'$  by  $y$  in the expression  $xy'$ , we could conclude that  $xy \in A$ . What we would like to do is assume that  $x, y \in A$ , then deduce

$$x, y' \in A, \quad (11)$$

and then conclude that

$$xy = xy'' \in A.$$

To deduce (11), we need to show if  $y \in A$ , then  $y' \in A$ . But this is (5). Detour! Let’s work on (5).

We will assume that  $y \in A$  and try to prove that  $y' \in A$ . If we could get rid of that pesky  $x$  (in the hypothesis  $x, y \in A$  implies  $xy' \in A$ ), we would have exactly what we want. We can effectively eliminate  $x$  by taking  $x = 1$  since  $1y = y$ . Formally, we argue as follows. Let  $y$  be in  $A$ . Since  $1 \in A$ ,  $y' = 1y' \in A$ . [ $y' = 1y'$  by Definition 11.3.1(b) and 11.3.1(d).] We have proved (5).

Now back to (4). Let  $x, y \in A$ . By the just proved (5),  $y' \in A$ . By the given condition,  $xy = xy'' \in A$ . [ $y = y''$  by Theorem 11.3.6(d).] We have proved that  $\cdot$  is a binary operator on  $A$ .

De Morgan's laws [Theorem 11.3.6(f)], in effect, allow us to interchange  $+$  and  $\cdot$ , so we can use them to prove that if  $x, y \in A$ , then  $x + y \in A$ . Formally we argue as follows. Suppose that  $x, y \in A$ . By (5),  $x'$  and  $y'$  are both in  $A$ . Since we have already proved that  $\cdot$  is a binary operator on  $A$ ,  $x'y' \in A$ . By (5),  $(x'y')' \in A$ . By De Morgan's laws [Theorem 11.3.6(f)] and Theorem 11.3.6(d),  $x+y = x''+y'' = (x'y')' \in A$ . Therefore,  $+$  is a binary operator on  $A$ . We have proved (4).

The next statement to prove is (6), which is to verify the associative laws

$$(x+y)+z = x+(y+z), \\ (xy)z = x(yz) \quad \text{for all } x, y, z \in A.$$

Now  $(S, +, \cdot, ', 0, 1)$  is a Boolean algebra and so the associative laws hold in  $S$ . Since  $A$  is a subset of  $S$ , the associative laws surely hold in  $A$ . Thus (6) holds. For the same reason, properties (7) through (10) also hold in  $A$ . Therefore,  $(A, +, \cdot, ', 0, 1)$  is a Boolean algebra.

### Formal Solution

Suppose that  $(A, +, \cdot, ', 0, 1)$  is a Boolean algebra. Then  $1 \in A$ . Suppose that  $x, y \in A$ . Then  $y' \in A$ . Thus  $xy' \in A$ .

Now suppose that  $1 \in A$  and  $xy' \in A$  for all  $x, y \in A$ . Taking  $x = y = 1$ , we obtain  $0 = 11' \in A$ . Taking  $x = 1$ , we obtain  $y' = 1y' \in A$ . Thus  $'$  is a unary operator on  $A$ . Replacing  $y$  by  $y'$ , we obtain  $xy = xy'' \in A$ . Thus  $\cdot$  is a binary operator on  $A$ . Now  $x + y = x'' + y'' = (x'y')' \in A$ . Thus  $+$  is a binary operator on  $A$ . Parts a–e of Definition 11.3.1 automatically hold in  $A$  since they hold in  $S$ . Therefore  $(A, +, \cdot, ', 0, 1)$  is a Boolean algebra.

### Summary of Problem-Solving Techniques

- When trying to construct a proof, write out carefully what is assumed and what is to be proved.
- When trying to construct a proof, look at closely related definitions and theorems.
- To prove that something is a Boolean algebra, go directly to the definition (Definition 11.3.1).
- Consider proving statements in an order different from that given. In this problem, it was easier to prove statement (5) before proving statement (4).
- Try various substitutions for the variables in a universally quantified statement. (After all, “universally quantified” means that the statement holds true *for all* values.) By taking  $x = y = 1$  in the statement

$$xy' \in A \quad \text{for all } x, y \in A,$$

we were able to prove that  $0 \in A$ .

### Exercises

Use the following definitions for Exercises 1–8. For  $a < b$ , let

$$[a, b) = \{x \in \mathbf{R} \mid a \leq x < b\},$$

where, as usual,  $\mathbf{R}$  denotes the set of real numbers. Define a subset  $X$  of  $\mathbf{R}$  to be open relative to the  $[a, b)$  definition if for every  $x \in X$ , there exists  $a, b \in \mathbf{R}$  with  $a < b$  and  $x \in [a, b) \subseteq X$ . Henceforth, we abbreviate “open relative to the  $[a, b)$  definition” to “open.” Let  $\mathbf{R}$  be the universal set. Define a set  $X \subseteq \mathbf{R}$  to be clopen if  $X$  and  $\bar{X}$  are both open. Let  $\mathcal{A}$  be the set of clopen subsets of  $\mathbf{R}$ .

1. Prove that  $\mathbf{R}$  and  $\emptyset$  are in  $\mathcal{A}$ .
2. Prove that  $[1, 2) \cup [3, 5) \in \mathcal{A}$ .
3. Prove that  $X = \{x \in \mathbf{R} \mid 3 < x < 10\}$  is open but not clopen.
4. Prove that if  $X \in \mathcal{A}$ , then  $\bar{X} \in \mathcal{A}$ .
5. Prove that if  $X$  and  $Y$  are open, then  $X \cup Y$  is open.
6. Prove that if  $X$  and  $Y$  are open, then  $X \cap Y$  is open.
7. Prove that if  $X, Y \in \mathcal{A}$ , then  $X \cap Y \in \mathcal{A}$ .
8. Prove that  $(\mathcal{A}, \cup, \cap, \bar{\phantom{x}}, \emptyset, \mathbf{R})$  is a Boolean algebra.
9. Can we prove Exercises 1 and 4–8 if we replace  $[a, b)$ ,  $a < b$ , by  $\{x \in \mathbf{R} \mid a \leq x \leq b\}$ ?

## 11.4 Boolean Functions and Synthesis of Circuits

A circuit is constructed to carry out a specified task. If we want to construct a combinatorial circuit, the problem can be given in terms of inputs and outputs. For example, suppose that we want to construct a combinatorial circuit to compute the **exclusive-OR** of  $x_1$  and  $x_2$ . We can state the problem by listing the inputs and outputs that define the exclusive-OR. This is equivalent to giving the desired logic table.

**Definition 11.4.1** ► The *exclusive-OR* of  $x_1$  and  $x_2$  written  $x_1 \oplus x_2$  is defined by Table 11.4.1.

**TABLE 11.4.1** ■ The exclusive-OR

| $x_1$ | $x_2$ | $x_1 \oplus x_2$ |
|-------|-------|------------------|
| 1     | 1     | 0                |
| 1     | 0     | 1                |
| 0     | 1     | 1                |
| 0     | 0     | 0                |

A logic table, with one output, is a function. The domain is the set of inputs and the range is the set of outputs. For the exclusive-OR function given in Table 11.4.1, the domain is the set  $\{(1, 1), (1, 0), (0, 1), (0, 0)\}$  and the range is the set  $Z_2 = \{0, 1\}$ .

If we could develop a formula for the exclusive-OR function of the form

$$x_1 \oplus x_2 = X(x_1, x_2),$$

where  $X$  is a Boolean expression, we could solve the problem of constructing the combinatorial circuit. We could merely construct the circuit corresponding to  $X$ .

Functions that can be represented by Boolean expressions are called **Boolean functions**.

**Definition 11.4.2** ► Let  $X(x_1, \dots, x_n)$  be a Boolean expression. A function  $f$  of the form

$$f(x_1, \dots, x_n) = X(x_1, \dots, x_n)$$

is called a *Boolean function*.

### Example 11.4.3

The function  $f: Z_2^3 \rightarrow Z_2$  defined by

$$f(x_1, x_2, x_3) = x_1 \wedge (\bar{x}_2 \vee x_3)$$

is a Boolean function. The inputs and outputs are given in the following table.

| $x_1$ | $x_2$ | $x_3$ | $f(x_1, x_2, x_3)$ |
|-------|-------|-------|--------------------|
| 1     | 1     | 1     | 1                  |
| 1     | 1     | 0     | 0                  |
| 1     | 0     | 1     | 1                  |
| 1     | 0     | 0     | 1                  |
| 0     | 1     | 1     | 0                  |
| 0     | 1     | 0     | 0                  |
| 0     | 0     | 1     | 0                  |
| 0     | 0     | 0     | 0                  |

In the next example we show how an arbitrary function  $f: Z_2^n \rightarrow Z_2$  can be realized as a Boolean function.

**Example 11.4.4** Show that the function  $f$  given by the following table is a Boolean function.

| $x_1$ | $x_2$ | $x_3$ | $f(x_1, x_2, x_3)$ |
|-------|-------|-------|--------------------|
| 1     | 1     | 1     | 1                  |
| 1     | 1     | 0     | 0                  |
| 1     | 0     | 1     | 0                  |
| 1     | 0     | 0     | 1                  |
| 0     | 1     | 1     | 0                  |
| 0     | 1     | 0     | 1                  |
| 0     | 0     | 1     | 0                  |
| 0     | 0     | 0     | 0                  |

**SOLUTION** Consider the first row of the table and the combination

$$x_1 \wedge x_2 \wedge x_3. \quad (11.4.1)$$

Notice that if  $x_1 = x_2 = x_3 = 1$ , as indicated in the first row of the table, then (11.4.1) is 1. The values of  $x_i$  given by any other row of the table give (11.4.1) the value 0. Similarly, for the fourth row of the table we may construct the combination

$$x_1 \wedge \bar{x}_2 \wedge \bar{x}_3. \quad (11.4.2)$$

Expression (11.4.2) has the value 1 for the values of  $x_i$  given by the fourth row of the table, whereas the values of  $x_i$  given by any other row of the table give (11.4.2) the value 0.

The procedure is clear. We consider a row  $R$  of the table where the output is 1. We then form the combination  $x_1 \wedge x_2 \wedge x_3$  and place a bar over each  $x_i$  whose value is 0 in row  $R$ . The combination formed is 1 if and only if the  $x_i$  have the values given in row  $R$ . Thus, for row 6, we obtain the combination

$$\bar{x}_1 \wedge x_2 \wedge \bar{x}_3. \quad (11.4.3)$$

Next, we OR the terms (11.4.1)–(11.4.3) to obtain the Boolean expression

$$(x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \bar{x}_2 \wedge \bar{x}_3) \vee (\bar{x}_1 \wedge x_2 \wedge \bar{x}_3). \quad (11.4.4)$$

We claim that  $f(x_1, x_2, x_3)$  and (11.4.4) are equal. To verify this, first suppose that  $x_1, x_2$ , and  $x_3$  have values given by a row of the table for which  $f(x_1, x_2, x_3) = 1$ . Then one of (11.4.1)–(11.4.3) is 1, so the value of (11.4.4) is 1. On the other hand, if  $x_1, x_2, x_3$  have values given by a row of the table for which  $f(x_1, x_2, x_3) = 0$ , all of (11.4.1)–(11.4.3) are 0, so the value of (11.4.4) is 0. Thus  $f$  and the Boolean expression (11.4.4) agree on  $Z_2^3$ ; therefore,

$$f(x_1, x_2, x_3) = (x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \bar{x}_2 \wedge \bar{x}_3) \vee (\bar{x}_1 \wedge x_2 \wedge \bar{x}_3),$$

as claimed. ◀

After one more definition, we will show that the method of Example 11.4.4 can be used to represent any function  $f: Z_2^n \rightarrow Z_2$ .

**Definition 11.4.5** ► A *minterm* in the symbols  $x_1, \dots, x_n$  is a Boolean expression of the form

$$y_1 \wedge y_2 \wedge \cdots \wedge y_n,$$

where each  $y_i$  is either  $x_i$  or  $\bar{x}_i$ . ◀

**Theorem 11.4.6**

If  $f: Z_2^n \rightarrow Z_2$ , then  $f$  is a Boolean function. If  $f$  is not identically zero, let  $A_1, \dots, A_k$  denote the elements  $A_i$  of  $Z_2^n$  for which  $f(A_i) = 1$ . For each  $A_i = (a_1, \dots, a_n)$ , set

$$m_i = y_1 \wedge \cdots \wedge y_n,$$

where

$$y_j = \begin{cases} x_j & \text{if } a_j = 1 \\ \bar{x}_j & \text{if } a_j = 0. \end{cases}$$

Then

$$f(x_1, \dots, x_n) = m_1 \vee m_2 \vee \cdots \vee m_k. \quad (11.4.5)$$

**Proof** If  $f(x_1, \dots, x_n) = 0$  for all  $x_i$ , then  $f$  is a Boolean function, since 0 is a Boolean expression.

Suppose that  $f$  is not identically zero. Let  $m_i(a_1, \dots, a_n)$  denote the value obtained from  $m_i$  by replacing each  $x_j$  with  $a_j$ . It follows from the definition of  $m_i$  that

$$m_i(A) = \begin{cases} 1 & \text{if } A = A_i \\ 0 & \text{if } A \neq A_i. \end{cases}$$

Let  $A \in Z_2^n$ . If  $A = A_i$  for some  $i \in \{1, \dots, k\}$ , then  $f(A) = 1$ ,  $m_i(A) = 1$ , and

$$m_1(A) \vee \cdots \vee m_k(A) = 1.$$

On the other hand, if  $A \neq A_i$  for any  $i \in \{1, \dots, k\}$ , then  $f(A) = 0$ ,  $m_i(A) = 0$  for  $i = 1, \dots, k$ , and

$$m_1(A) \vee \cdots \vee m_k(A) = 0.$$

Therefore, (11.4.5) holds. ◀

**Definition 11.4.7** ► The representation (11.4.5) of a Boolean function  $f: Z_2^n \rightarrow Z_2$  is called the *disjunctive normal form* of the function  $f$ . ◀

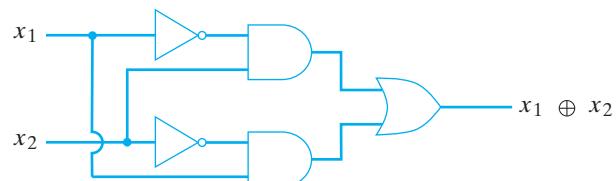
**Example 11.4.8**

Design a combinatorial circuit that computes the exclusive-OR of  $x_1$  and  $x_2$ .

**SOLUTION** The logic table for the exclusive-OR function  $x_1 \oplus x_2$  is given in Table 11.4.1. The disjunctive normal form of this function is

$$x_1 \oplus x_2 = (x_1 \wedge \bar{x}_2) \vee (\bar{x}_1 \wedge x_2). \quad (11.4.6)$$

The combinatorial circuit corresponding to (11.4.6) is given in Figure 11.4.1.



**Figure 11.4.1** A combinatorial circuit for the exclusive-OR.

Suppose that a function is given by a Boolean expression such as

$$f(x_1, x_2, x_3) = (x_1 \vee x_2) \wedge x_3$$

and we wish to find the disjunctive normal form of  $f$ . We could write the logic table for  $f$  and then use Theorem 11.4.6. Alternatively, we can deal directly with the Boolean expression by using the definitions and results of Sections 11.2 and 11.3. We begin by distributing the term  $x_3$  as follows:

$$(x_1 \vee x_2) \wedge x_3 = (x_1 \wedge x_3) \vee (x_2 \wedge x_3).$$

Although this represents the Boolean expression as a combination of terms of the form  $y \wedge z$ , it is not in disjunctive normal form, since each term does not contain all of the symbols  $x_1$ ,  $x_2$ , and  $x_3$ . However, this is easily remedied, as follows:

$$\begin{aligned} (x_1 \wedge x_3) \vee (x_2 \wedge x_3) &= (x_1 \wedge x_3 \wedge 1) \vee (x_2 \wedge x_3 \wedge 1) \\ &= (x_1 \wedge x_3 \wedge (x_2 \vee \bar{x}_2)) \vee (x_2 \wedge x_3 \wedge (x_1 \vee \bar{x}_1)) \\ &= (x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \bar{x}_2 \wedge x_3) \\ &\quad \vee (x_1 \wedge x_2 \wedge x_3) \vee (\bar{x}_1 \wedge x_2 \wedge x_3) \\ &= (x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \bar{x}_2 \wedge x_3) \\ &\quad \vee (\bar{x}_1 \wedge x_2 \wedge x_3). \end{aligned}$$

This expression is the disjunctive normal form of  $f$ .

Theorem 11.4.6 has a dual. In this case the function  $f$  is expressed as

$$f(x_1, \dots, x_n) = M_1 \wedge M_2 \wedge \dots \wedge M_k. \quad (11.4.7)$$

Each  $M_i$  is of the form

$$y_1 \vee \dots \vee y_n, \quad (11.4.8)$$

### Go Online

For more on conjunctive normal form, see  
[goo.gl/5HWh0y](http://goo.gl/5HWh0y)

where  $y_j$  is either  $x_j$  or  $\bar{x}_j$ . A term of the form (11.4.8) is called a **maxterm** and the representation of  $f$  (11.4.7) is called the **conjunctive normal form**. Exercises 24–28 explore maxterms and the conjunctive normal form in more detail.

## 11.4 Review Exercises

1. Define *exclusive-OR*.
2. What is a Boolean function?
3. What is a minterm?
4. What is the disjunctive normal form of a Boolean function?
5. How can one obtain the disjunctive normal form of a Boolean function?
6. What is a maxterm?
7. What is the conjunctive normal form of a Boolean function?

## 11.4 Exercises

In Exercises 1–10, find the disjunctive normal form of each function and draw the combinatorial circuit corresponding to the disjunctive normal form.

| 1. | $x$ | $y$ | $f(x, y)$ |
|----|-----|-----|-----------|
|    | 1   | 1   | 1         |
|    | 1   | 0   | 0         |
|    | 0   | 1   | 1         |
|    | 0   | 0   | 1         |

| 2. | $x$ | $y$ | $f(x, y)$ |
|----|-----|-----|-----------|
|    | 1   | 1   | 0         |
|    | 1   | 0   | 1         |
|    | 0   | 1   | 0         |
|    | 0   | 0   | 1         |

3.

| $x$ | $y$ | $z$ | $f(x, y, z)$ |
|-----|-----|-----|--------------|
| 1   | 1   | 1   | 1            |
| 1   | 1   | 0   | 1            |
| 1   | 0   | 1   | 0            |
| 1   | 0   | 0   | 1            |
| 0   | 1   | 1   | 0            |
| 0   | 1   | 0   | 0            |
| 0   | 0   | 1   | 1            |
| 0   | 0   | 0   | 1            |

4.

| $x$ | $y$ | $z$ | $f(x, y, z)$ |
|-----|-----|-----|--------------|
| 1   | 1   | 1   | 1            |
| 1   | 1   | 0   | 1            |
| 1   | 0   | 1   | 0            |
| 1   | 0   | 0   | 1            |
| 0   | 1   | 1   | 1            |
| 0   | 1   | 0   | 1            |
| 0   | 0   | 1   | 0            |
| 0   | 0   | 0   | 0            |

5.

| $x$ | $y$ | $z$ | $f(x, y, z)$ |
|-----|-----|-----|--------------|
| 1   | 1   | 1   | 0            |
| 1   | 1   | 0   | 1            |
| 1   | 0   | 1   | 1            |
| 1   | 0   | 0   | 1            |
| 0   | 1   | 1   | 1            |
| 0   | 1   | 0   | 1            |
| 0   | 0   | 1   | 1            |
| 0   | 0   | 0   | 0            |

6.

| $x$ | $y$ | $z$ | $f(x, y, z)$ |
|-----|-----|-----|--------------|
| 1   | 1   | 1   | 1            |
| 1   | 1   | 0   | 1            |
| 1   | 0   | 1   | 1            |
| 1   | 0   | 0   | 0            |
| 0   | 1   | 1   | 0            |
| 0   | 1   | 0   | 1            |
| 0   | 0   | 1   | 1            |
| 0   | 0   | 0   | 1            |

7.

| $x$ | $y$ | $z$ | $f(x, y, z)$ |
|-----|-----|-----|--------------|
| 1   | 1   | 1   | 1            |
| 1   | 1   | 0   | 0            |
| 1   | 0   | 1   | 0            |
| 1   | 0   | 0   | 1            |
| 0   | 1   | 1   | 0            |
| 0   | 1   | 0   | 0            |
| 0   | 0   | 1   | 0            |
| 0   | 0   | 0   | 1            |

8.

| $x$ | $y$ | $z$ | $f(x, y, z)$ |
|-----|-----|-----|--------------|
| 1   | 1   | 1   | 0            |
| 1   | 1   | 0   | 0            |
| 1   | 0   | 1   | 0            |
| 1   | 0   | 0   | 1            |
| 0   | 1   | 1   | 1            |
| 0   | 1   | 0   | 1            |
| 0   | 0   | 1   | 1            |
| 0   | 0   | 0   | 0            |

9.

| $w$ | $x$ | $y$ | $z$ | $f(w, x, y, z)$ |
|-----|-----|-----|-----|-----------------|
| 1   | 1   | 1   | 1   | 1               |
| 1   | 1   | 1   | 0   | 0               |
| 1   | 1   | 0   | 1   | 1               |
| 1   | 1   | 0   | 0   | 0               |
| 1   | 0   | 1   | 1   | 0               |
| 1   | 0   | 1   | 0   | 0               |
| 1   | 0   | 0   | 1   | 1               |
| 1   | 0   | 0   | 0   | 0               |
| 0   | 1   | 1   | 1   | 1               |
| 0   | 1   | 1   | 0   | 0               |
| 0   | 1   | 0   | 1   | 0               |
| 0   | 1   | 0   | 0   | 0               |
| 0   | 0   | 1   | 1   | 1               |
| 0   | 0   | 1   | 0   | 0               |
| 0   | 0   | 0   | 1   | 0               |
| 0   | 0   | 0   | 0   | 0               |

10.

| $w$ | $x$ | $y$ | $z$ | $f(w, x, y, z)$ |
|-----|-----|-----|-----|-----------------|
| 1   | 1   | 1   | 1   | 0               |
| 1   | 1   | 1   | 0   | 0               |
| 1   | 1   | 0   | 1   | 1               |
| 1   | 1   | 0   | 0   | 1               |
| 1   | 0   | 1   | 1   | 1               |
| 1   | 0   | 1   | 0   | 1               |
| 1   | 0   | 0   | 1   | 0               |
| 1   | 0   | 0   | 0   | 1               |
| 0   | 1   | 1   | 1   | 0               |
| 0   | 1   | 1   | 0   | 1               |
| 0   | 1   | 0   | 1   | 1               |
| 0   | 1   | 0   | 0   | 1               |
| 0   | 0   | 1   | 1   | 0               |
| 0   | 0   | 1   | 0   | 1               |
| 0   | 0   | 0   | 1   | 0               |
| 0   | 0   | 0   | 0   | 1               |

In Exercises 11–20, find the disjunctive normal form of each function using algebraic techniques. (We abbreviate  $a \wedge b$  as  $ab$ .)

$$11. f(x, y) = x \vee xy$$

$$12. f(x, y) = (x \vee y)(\bar{x} \vee \bar{y})$$

$$13. f(x, y, z) = x \vee y(x \vee \bar{z})$$

$$14. f(x, y, z) = (yz \vee x\bar{z})(\bar{x}\bar{y} \vee z)$$

$$15. f(x, y, z) = x \vee (\bar{y} \vee (x\bar{y} \vee x\bar{z}))$$

$$16. f(x, y, z) = (\bar{x}y \vee \bar{x}\bar{z})(\bar{x} \vee yz)$$

$$17. f(x, y, z) = (x \vee \bar{x}y \vee \bar{x}\bar{y}\bar{z})(xy \vee \bar{x}z)(y \vee xy\bar{z})$$

$$18. f(w, x, y, z) = wy \vee (w\bar{y} \vee z)(x \vee \bar{w}z)$$

$$19. f(x, y, z) = (\bar{x}y \vee \bar{x}\bar{z})(xyz \vee y\bar{z})(x\bar{y}z \vee x\bar{y}\bar{z} \vee x\bar{y}z \vee x\bar{y}\bar{z})$$

$$20. f(w, x, y, z) = (\bar{w}\bar{x}\bar{y}z \vee \bar{x}\bar{y}\bar{z})(\bar{w}\bar{y}z \vee xy\bar{z} \vee yxz)(\bar{w}z \vee xy \vee \bar{w}\bar{y}z \vee xy\bar{z} \vee \bar{x}yz)$$

21. How many Boolean functions are there from  $Z_2^n$  into  $Z_2$ ?

Let  $F$  denote the set of all functions from  $Z_2^n$  into  $Z_2$ . Define

$$\begin{aligned} (f \vee g)(x) &= f(x) \vee g(x) & x \in Z_2^n \\ (f \wedge g)(x) &= f(x) \wedge g(x) & x \in Z_2^n \\ \bar{f}(x) &= \bar{f}(x) & x \in Z_2^n \\ 0(x) &= 0 & x \in Z_2^n \\ 1(x) &= 1 & x \in Z_2^n \end{aligned}$$

- 22. How many elements does  $F$  have?
- 23. Show that  $(F, \vee, \wedge, \neg, 0, 1)$  is a Boolean algebra.
- 24. By dualizing the procedure of Example 11.4.4, explain how to find the conjunctive normal form of a Boolean function from  $Z_2^n$  into  $Z_2$ .
- 25. Find the conjunctive normal form of each function in Exercises 1–10.

- 26. By using algebraic methods, find the conjunctive normal form of each function in Exercises 11–20.
- 27. Show that if  $m_1 \vee \dots \vee m_k$  is the disjunctive normal form of  $f(x_1, \dots, x_n)$ , then  $\bar{m}_1 \wedge \dots \wedge \bar{m}_k$  is the conjunctive normal form of  $\bar{f}(x_1, \dots, x_n)$ .
- 28. Using the method of Exercise 27, find the conjunctive normal form of  $\bar{f}$  for each function  $f$  of Exercises 1–10.
- 29. Show that the disjunctive normal form (11.4.5) is unique; that is, show that if we have a Boolean function

$$f(x_1, \dots, x_n) = m_1 \vee \dots \vee m_k = m'_1 \vee \dots \vee m'_j,$$

where each  $m_i, m'_i$  is a minterm, then  $k = j$  and the subscripts on the  $m'_i$  may be permuted so that  $m_i = m'_i$  for  $i = 1, \dots, k$ .

## 11.5 Applications

### Go Online

For more on logic design, see [goo.gl/5HWh0y](#)

In the preceding section we showed how to design a combinatorial circuit using AND, OR, and NOT gates that would compute an arbitrary function from  $Z_2^n$  into  $Z_2$ , where  $Z_2 = \{0, 1\}$ . In this section we consider using other kinds of gates to implement a circuit. We also consider the problem of efficient design. We will conclude by looking at several useful circuits having multiple outputs. Throughout this section, we write  $ab$  for  $a \wedge b$ .

Before considering alternatives to AND, OR, and NOT gates, we must give a precise definition of “gate.”

**Definition 11.5.1** ► A *gate* is a function from  $Z_2^n$  into  $Z_2$ .

### Example 11.5.2

The AND gate is the function  $\wedge$  from  $Z_2^2$  into  $Z_2$  defined as in Definition 11.1.1. The NOT gate is the function  $\neg$  from  $Z_2$  into  $Z_2$  defined as in Definition 11.1.3.

We are interested in gates that allow us to construct arbitrary combinatorial circuits.

**Definition 11.5.3** ► A set of gates  $\{g_1, \dots, g_k\}$  is said to be *functionally complete* if, given any positive integer  $n$  and a function  $f$  from  $Z_2^n$  into  $Z_2$ , it is possible to construct a combinatorial circuit that computes  $f$  using only the gates  $g_1, \dots, g_k$ .

### Example 11.5.4

Theorem 11.4.6 shows that the set of gates {AND, OR, NOT} is functionally complete.

It is an interesting fact that we can eliminate either AND or OR from the set {AND, OR, NOT} and still obtain a functionally complete set of gates.

### Theorem 11.5.5

The sets of gates

$$\{\text{AND, NOT}\} \quad \{\text{OR, NOT}\}$$

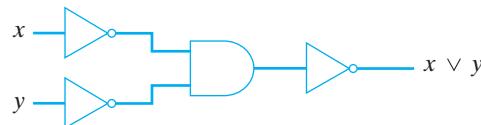
are functionally complete.

**Proof** We will show that the set of gates {AND, NOT} is functionally complete and leave the problem of showing that the other set is functionally complete for the exercises (see Exercise 1).

We have

$$\begin{aligned} x \vee y &= \bar{\bar{x}} \vee \bar{\bar{y}} && \text{involution law} \\ &= \bar{x} \bar{y} && \text{De Morgan's law.} \end{aligned}$$

Therefore, an OR gate can be replaced by one AND gate and three NOT gates. (The combinatorial circuit is shown in Figure 11.5.1.)



**Figure 11.5.1** A combinational circuit using only AND and NOT gates that computes  $x \vee y$ .

Given any function  $f: Z_2^n \rightarrow Z_2$ , by Theorem 11.4.6 we can construct a combinatorial circuit  $C$  using AND, OR, and NOT gates that computes  $f$ . But Figure 11.5.1 shows that each OR gate can be replaced by AND and NOT gates. Therefore, the circuit  $C$  can be modified so that it consists only of AND and NOT gates. Thus the set of gates {AND, NOT} is functionally complete. ◀

Although none of AND, OR, or NOT singly forms a functionally complete set (see Exercises 2–4), it is possible to define a new gate that by itself forms a functionally complete set.

**Definition 11.5.6** ► A *NAND gate* receives inputs  $x_1$  and  $x_2$ , where  $x_1$  and  $x_2$  are bits, and produces output denoted  $x_1 \uparrow x_2$ , where



**Figure 11.5.2** NAND gate. ◀

$$x_1 \uparrow x_2 = \begin{cases} 0 & \text{if } x_1 = 1 \text{ and } x_2 = 1 \\ 1 & \text{otherwise.} \end{cases}$$

A NAND gate is drawn as shown in Figure 11.5.2. ◀

Many basic circuits used in digital computers today are built from NAND gates.

### Theorem 11.5.7

The set {NAND} is a functionally complete set of gates.

**Proof** First we observe that  $x \uparrow y = \bar{x}\bar{y}$ . Therefore,

$$\bar{x} = \bar{x}\bar{x} = x \uparrow x \quad (11.5.1)$$

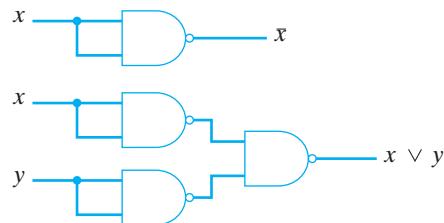
$$x \vee y = \bar{\bar{x}}\bar{\bar{y}} = \bar{x} \uparrow \bar{y} = (x \uparrow x) \uparrow (y \uparrow y). \quad (11.5.2)$$

Equations (11.5.1) and (11.5.2) show that both OR and NOT can be written in terms of NAND. By Theorem 11.5.5, the set {OR, NOT} is functionally complete. It follows that the set {NAND} is also functionally complete. ◀

### Example 11.5.8

Design combinatorial circuits using NAND gates to compute the functions  $f_1(x) = \bar{x}$  and  $f_2(x, y) = x \vee y$ .

**SOLUTION** The combinatorial circuits, derived from equations (11.5.1) and (11.5.2), are shown in Figure 11.5.3.



**Figure 11.5.3** Combinatorial circuits using only NAND gates that compute  $\bar{x}$  and  $x \vee y$ .

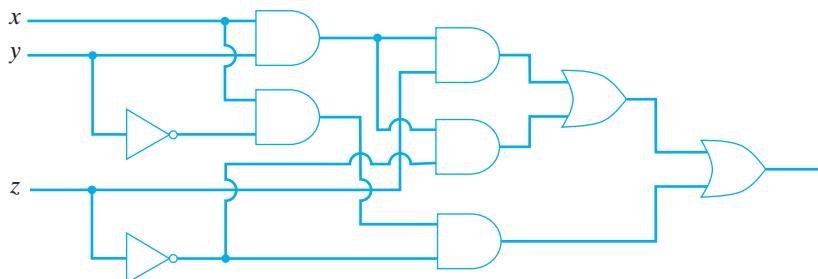
Consider the problem of designing a combinatorial circuit using AND, OR, and NOT gates to compute the function  $f$ .

| $x$ | $y$ | $z$ | $f(x, y, z)$ |
|-----|-----|-----|--------------|
| 1   | 1   | 1   | 1            |
| 1   | 1   | 0   | 1            |
| 1   | 0   | 1   | 0            |
| 1   | 0   | 0   | 1            |
| 0   | 1   | 1   | 0            |
| 0   | 1   | 0   | 0            |
| 0   | 0   | 1   | 0            |
| 0   | 0   | 0   | 0            |

The disjunctive normal form of  $f$  is

$$f(x, y, z) = xyz \vee xy\bar{z} \vee x\bar{y}\bar{z}. \quad (11.5.3)$$

The combinatorial circuit corresponding to (11.5.3) is shown in Figure 11.5.4.



**Figure 11.5.4** A combinatorial circuit that computes  $f(x, y, z) = xyz \vee xy\bar{z} \vee x\bar{y}\bar{z}$ .

The circuit in Figure 11.5.4 has nine gates. As we will show, it is possible to design a circuit having fewer gates. The problem of finding the best circuit is called the **minimization problem**. There are many definitions of “best.”

To find a simpler combinatorial circuit equivalent to that in Figure 11.5.4, we attempt to simplify the Boolean expression (11.5.3) that represents it. The equations

$$Ea \vee E\bar{a} = E \quad (11.5.4)$$

$$E = E \vee Ea, \quad (11.5.5)$$

where  $E$  represents an arbitrary Boolean expression, are useful in simplifying Boolean expressions.

Equation (11.5.4) may be derived as follows:

$$Ea \vee E\bar{a} = E(a \vee \bar{a}) = E1 = E$$

using the properties of Boolean algebras. Equation (11.5.5) is essentially the absorption law [Theorem 11.3.6(c)].

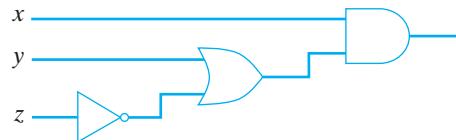
Using (11.5.4) and (11.5.5), we may simplify (11.5.3) as follows:

$$\begin{aligned} xyz \vee xy\bar{z} \vee x\bar{y}\bar{z} &= xy \vee x\bar{y}\bar{z} && \text{by (11.5.4)} \\ &= xy \vee xy\bar{z} \vee x\bar{y}\bar{z} && \text{by (11.5.5)} \\ &= xy \vee x\bar{z}. && \text{by (11.5.4).} \end{aligned}$$

A further simplification,

$$xy \vee x\bar{z} = x(y \vee \bar{z}), \quad (11.5.6)$$

is possible using the distributive law [Definition 11.3.1(c)]. The combinatorial circuit corresponding to (11.5.6), which requires only three gates, is shown in Figure 11.5.5.

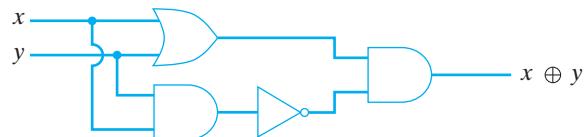


**Figure 11.5.5** A three-gate combinatorial circuit equivalent to that of Figure 11.5.4.

### Example 11.5.9

The combinatorial circuit in Figure 11.4.1 uses five AND, OR, and NOT gates to compute the exclusive-OR  $x \oplus y$  of  $x$  and  $y$ . Design a circuit that computes  $x \oplus y$  using fewer AND, OR, and NOT gates.

**SOLUTION** Unfortunately, (11.5.4) and (11.5.5) do not help us simplify the disjunctive normal form  $x\bar{y} \vee \bar{x}y$  of  $x \oplus y$ . Thus we must experiment with various Boolean rules until we produce an expression that requires fewer than five gates. One solution is provided by the expression  $(x \vee y)\bar{x}\bar{y}$  whose implementation requires only four gates. This combinatorial circuit is shown in Figure 11.5.6.



**Figure 11.5.6** A four-gate combinatorial circuit that computes the exclusive-OR  $x \oplus y$  of  $x$  and  $y$ .

The set of gates available determines the minimization problem. Since the state of technology determines the available gates, the minimization problem changes through time. In the 1950s, the typical problem was to minimize circuits consisting of AND, OR, and NOT gates. Solutions such as the Quine–McCluskey method and the method

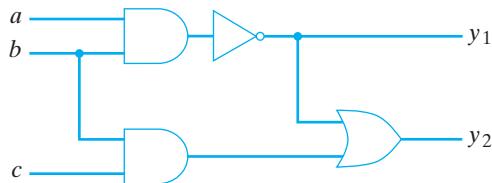
of Karnaugh maps were provided. The reader is referred to [Mendelson] for the details of these methods.

Advances in solid-state technology have made it possible to manufacture very small components, called **integrated circuits**, which are themselves entire circuits. Thus circuit design today consists of combining basic gates such as AND, OR, NOT, and NAND gates and integrated circuits to compute the desired functions. Boolean algebra remains an essential tool, as a glance at a book on logic design such as [McCalla] will show.

We conclude this section by considering several useful combinatorial circuits having multiple outputs. A circuit with  $n$  outputs can be characterized by  $n$  Boolean expressions, as the next example shows.

**Example 11.5.10** Write two Boolean expressions to describe the combinatorial circuit of Figure 11.5.7.

**SOLUTION** The output  $y_1$  is described by the expression  $y_1 = \overline{ab}$ , and  $y_2$  is described by the expression  $y_2 = bc \vee \overline{ab}$ .



**Figure 11.5.7** A combinational circuit with two outputs.

Our first circuit is called a **half adder**.

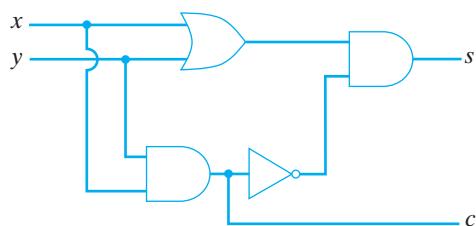
**Definition 11.5.11** ► A *half adder* accepts as input two bits  $x$  and  $y$  and produces as output the binary sum  $cs$  of  $x$  and  $y$ . The term  $cs$  is a two-bit binary number. We call  $s$  the sum bit and  $c$  the carry bit.

**Example 11.5.12** **Half-Adder Circuit** Design a half-adder combinatorial circuit.

**SOLUTION** The table for the half-adder circuit is as follows:

| $x$ | $y$ | $c$ | $s$ |
|-----|-----|-----|-----|
| 1   | 1   | 1   | 0   |
| 1   | 0   | 0   | 1   |
| 0   | 1   | 0   | 1   |
| 0   | 0   | 0   | 0   |

This function has two outputs,  $c$  and  $s$ . We observe that  $c = xy$  and  $s = x \oplus y$ . Thus we obtain the half-adder circuit of Figure 11.5.8. We used the circuit of Figure 11.5.6 to realize the exclusive-OR.



**Figure 11.5.8** A half-adder circuit.

A **full adder** sums three bits and is useful for adding two bits and a third carry bit from a previous addition.

**Definition 11.5.13** ► A *full adder* accepts as input three bits  $x$ ,  $y$ , and  $z$  and produces as output the binary sum  $cs$  of  $x$ ,  $y$ , and  $z$ . The term  $cs$  is a two-bit binary number. ◀

**Example 11.5.14** **Full-Adder Circuit** Design a full-adder combinatorial circuit.

**SOLUTION** The table for the full-adder circuit is as follows:

| $x$ | $y$ | $z$ | $c$ | $s$ |
|-----|-----|-----|-----|-----|
| 1   | 1   | 1   | 1   | 1   |
| 1   | 1   | 0   | 1   | 0   |
| 1   | 0   | 1   | 1   | 0   |
| 1   | 0   | 0   | 0   | 1   |
| 0   | 1   | 1   | 1   | 0   |
| 0   | 1   | 0   | 0   | 1   |
| 0   | 0   | 1   | 0   | 1   |
| 0   | 0   | 0   | 0   | 0   |

Checking the eight possibilities, we see that  $s = x \oplus y \oplus z$ ; hence we can use two exclusive-OR circuits to compute  $s$ .

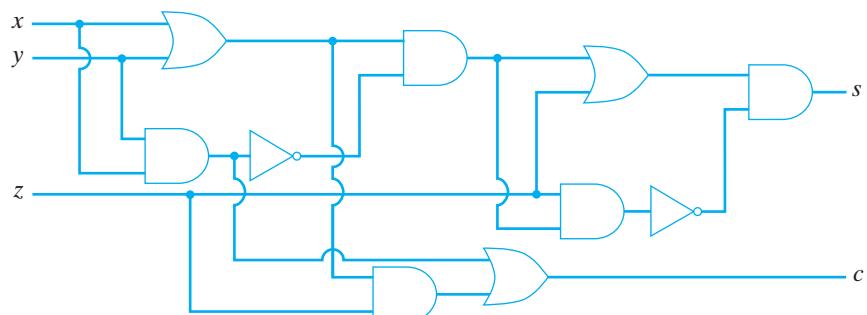
To compute  $c$ , we first find the disjunctive normal form

$$c = xyz \vee xy\bar{z} \vee x\bar{y}z \vee \bar{x}yz \quad (11.5.7)$$

of  $c$ . Next, we use (11.5.4) and (11.5.5) to simplify (11.5.7) as follows:

$$\begin{aligned} xyz \vee xy\bar{z} \vee x\bar{y}z \vee \bar{x}yz &= xy \vee x\bar{y}z \vee \bar{x}yz \\ &= xy \vee xyz \vee x\bar{y}z \vee \bar{x}yz \\ &= xy \vee xz \vee \bar{x}yz \\ &= xy \vee xz \vee xyz \vee \bar{x}yz \\ &= xy \vee xz \vee yz. \end{aligned}$$

Additional gates can be eliminated by writing  $c = xy \vee z(x \vee y)$ . We obtain the full-adder circuit given in Figure 11.5.9.

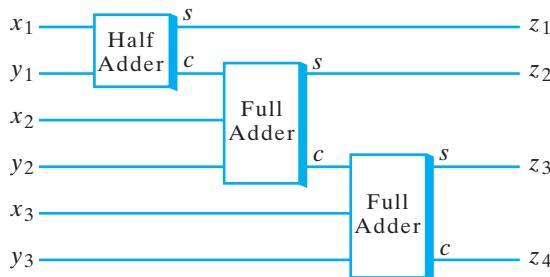


**Figure 11.5.9** A full-adder circuit. ◀

Our last example shows how we may use half-adder and full-adder circuits to construct a circuit to add binary numbers.

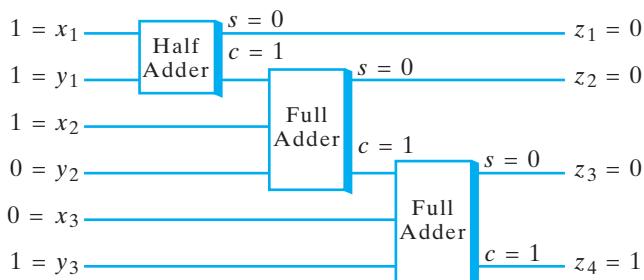
**Example 11.5.15 A Circuit to Add Binary Numbers** Using half-adder and full-adder circuits, design a combinatorial circuit that computes the sum of two three-bit numbers.

**SOLUTION** We will let  $M = x_3x_2x_1$  and  $N = y_3y_2y_1$  denote the numbers to be added and let  $z_4z_3z_2z_1$  denote the sum. The circuit that computes the sum of  $M$  and  $N$  is drawn in Figure 11.5.10. It is an implementation of the standard algorithm for adding numbers since the “carry bit” is indeed *carried* into the next binary addition.



**Figure 11.5.10** A combinatorial circuit that computes the sum of two three-bit numbers.

Figure 11.5.11 shows how the circuit adds  $011 = x_3x_2x_1$  and  $101 = y_3y_2y_1$ . The bits  $x_1 = 1$  and  $y_1 = 1$  are input to the half adder. Since  $1 + 1 = 10$ , the sum bit  $s = 0 = z_1$  and the carry bit  $c = 1$ . Next the carry bit (1) and the bits  $x_2 = 1$  and  $y_2 = 0$  are input to the (middle) full adder. Since  $1 + 1 + 0 = 10$ , the sum bit  $s = 0 = z_2$  and the carry bit  $c = 1$ . At the last step, the carry bit (1) and the bits  $x_3 = 0$  and  $y_3 = 1$  are input to the (bottom) full adder. Since  $1 + 0 + 1 = 10$ , the sum bit  $s = 0 = z_3$  and the carry bit  $c = 1 = z_4$ . The sum  $011 + 101 = 1000 = z_4z_3z_2z_1$ .



**Figure 11.5.11** The combinatorial circuit of Figure 11.5.10 adding  $011 = x_3x_2x_1$  and  $101 = y_3y_2y_1$ . The sum is  $1000 = z_4z_3z_2z_1$ .

If we were using three-bit registers for addition, so that the sum of two three-bit numbers would have to be no more than three bits, we could use the  $z_4$  bit in Example 11.5.15 as an overflow flag. If  $z_4 = 1$ , overflow occurred; if  $z_4 = 0$ , there was no overflow.

In the next chapter (Example 12.1.3), we discuss a sequential circuit that makes use of a primitive internal memory to add binary numbers.

## 11.5 Review Exercises

1. What is a gate?
2. What is a functionally complete set of gates?
3. Give examples of functionally complete sets of gates.
4. What is a NAND gate?
5. Is the set {NAND} functionally complete?
6. What is the minimization problem?
7. What is an integrated circuit?
8. Describe a half-adder circuit.
9. Describe a full-adder circuit.

## 11.5 Exercises

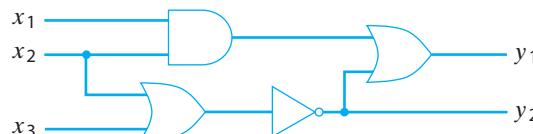
1. Show that the set of gates {OR, NOT} is functionally complete.

Show that each set of gates in Exercises 2–5 is not functionally complete.

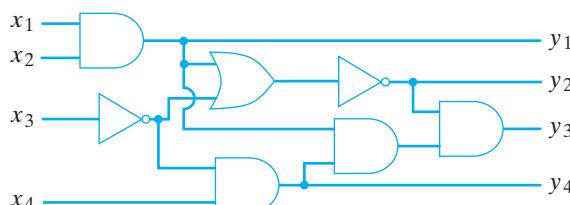
2. {AND}
3. {NOT}
4. {OR}
5. {AND, OR}
6. Draw a circuit using only NAND gates that computes  $xy$ .
7. Write  $xy$  using only  $\uparrow$ .
8. Prove or disprove:  $x \uparrow (y \uparrow z) = (x \uparrow y) \uparrow z$ , for all  $x, y, z \in Z_2$ .

Write Boolean expressions to describe the multiple output circuits in Exercises 9–11.

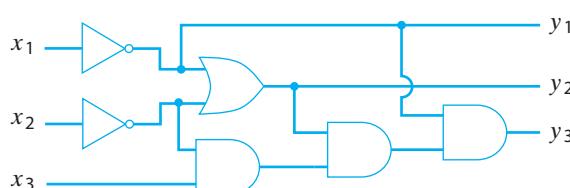
9.



10.



11.



12. Design circuits using only NAND gates to compute the functions of Exercises 1–10, Section 11.4.
13. Can you reduce the number of NAND gates used in any of your circuits for Exercise 12?
14. Design circuits using as few AND, OR, and NOT gates as you can to compute the functions of Exercises 1–10, Section 11.4.
15. Design a half-adder circuit using only NAND gates.
- ★16. Design a half-adder circuit using five NAND gates.

A NOR gate receives inputs  $x_1$  and  $x_2$ , where  $x_1$  and  $x_2$  are bits, and produces output denoted  $x_1 \downarrow x_2$ , where

$$x_1 \downarrow x_2 = \begin{cases} 0 & \text{if } x_1 = 1 \text{ or } x_2 = 1 \\ 1 & \text{otherwise.} \end{cases}$$

17. Write  $xy$ ,  $x \vee y$ ,  $\bar{x}$ , and  $x \uparrow y$  in terms of  $\downarrow$ .
18. Write  $x \downarrow y$  in terms of  $\uparrow$ .
19. Write the logic table for the NOR function.
20. Show that the set of gates {NOR} is functionally complete.
21. Design circuits using only NOR gates to compute the functions of Exercises 1–10, Section 11.4.
22. Can you reduce the number of NOR gates used in any of your circuits for Exercise 21?
23. Design a half-adder circuit using only NOR gates.
- ★24. Design a half-adder circuit using five NOR gates.
25. Design a circuit with three inputs that outputs 1 precisely when two or three inputs have value 1.
26. Design a circuit that multiplies the binary numbers  $x_2x_1$  and  $y_2y_1$ . The output will be of the form  $z_4z_3z_2z_1$ .
27. A **2's module** is a circuit that accepts as input two bits  $b$  and FLAGIN and outputs bits  $c$  and FLAGOUT. If FLAGIN = 1, then  $c = \bar{b}$  and FLAGOUT = 1. If FLAGIN = 0 and  $b = 1$ , then FLAGOUT = 1. If FLAGIN = 0 and  $b = 0$ , then FLAGOUT = 0. If FLAGIN = 0, then  $c = b$ . Design a circuit to implement a 2's module.

The 2's complement of a binary number can be computed by using the following algorithm.

### Algorithm 11.5.16

#### Finding the 2's Complement

This algorithm computes the 2's complement  $C_N C_{N-1} \dots C_2 C_1$  of the binary number  $M = B_N B_{N-1} \dots B_2 B_1$ . The number  $M$  is scanned from right to left and the bits are copied until 1 is found. Thereafter, if  $B_i = 0$ , we set  $C_i = 1$  and if  $B_i = 1$ , we set  $C_i = 0$ . The flag  $F$  indicates whether a 1 has been found ( $F = \text{true}$ ) or not ( $F = \text{false}$ ).

Input:  $B_N B_{N-1} \dots B_1$

Output:  $C_N C_{N-1} \dots C_1$

```
two_complement(B) {
 F = false
 i = 1
 while ($\neg F \wedge i \leq N$) {
 Ci = Bi
 if ($B_i == 1$)
 F = true
 i = i + 1
 }
 while ($i \leq N$) {
 Ci = $B_i \oplus 1$
 i = i + 1
 }
 return C
}
```

Find the 2's complement of the numbers in Exercises 28–30 using Algorithm 11.5.16.

28. 101100      29. 11011      30. 011010110

31. Using 2's modules, design a circuit that computes the 2's complement  $y_3 y_2 y_1$  of the three-bit binary number  $x_3 x_2 x_1$ .

\*32. Let  $*$  be a binary operator on a set  $S$  containing 0 and 1. Write a set of axioms for  $*$ , modeled after rules that NAND satisfies, so that if we define

$$\begin{aligned}\bar{x} &= x * x \\ x \vee y &= (x * x) * (y * y) \\ x \wedge y &= (x * y) * (x * y),\end{aligned}$$

then  $(S, \vee, \wedge, \neg, 0, 1)$  is a Boolean algebra.

\*33. Let  $*$  be a binary operator on a set  $S$  containing 0 and 1. Write a set of axioms for  $*$ , modeled after rules that NOR satisfies, and definitions for  $\neg$ ,  $\vee$ , and  $\wedge$  so that  $(S, \vee, \wedge, \neg, 0, 1)$  is a Boolean algebra.

\*34. Show that  $\{\rightarrow\}$  is functionally complete (see Definition 1.3.3).

\*35. Let  $B(x, y)$  be a Boolean expression in the variables  $x$  and  $y$  that uses only the operator  $\leftrightarrow$  (see Definition 1.3.8).

- (a) Show that if  $B$  contains an even number of  $x$ 's, the values of  $B(\bar{x}, y)$  and  $B(x, y)$  are the same for all  $x$  and  $y$ .
- (b) Show that if  $B$  contains an odd number of  $x$ 's, the values of  $B(\bar{x}, y)$  and  $\overline{B(x, y)}$  are the same for all  $x$  and  $y$ .
- (c) Use parts (a) and (b) to show that  $\{\leftrightarrow\}$  is *not* functionally complete.

This exercise was contributed by Paul Pluznikov.

## Chapter 11 Notes

General references on Boolean algebras are [Hohn; and Mendelson]. [Mendelson] contains over 150 references on Boolean algebras and combinatorial circuits. Books on logic design include [Kohavi; McCalla; and Ward].

[Hailperin] gives a technical discussion of Boole's mathematics. Additional references are also provided. Boole's book, *The Laws of Thought*, has been reprinted (see [Boole]).

Because of our interest in applications of Boolean algebra, most of our discussion was limited to the Boolean algebra  $(Z_2, \vee, \wedge, \neg, 0, 1)$ . However, versions of most of our results remain valid for arbitrary, finite Boolean algebras.

**Boolean expressions** in the symbols  $x_1, \dots, x_n$  over an arbitrary Boolean algebra  $(S, +, \cdot, ', 0, 1)$  are defined recursively as

- For each  $s \in S$ ,  $s$  is a Boolean expression.
- $x_1, \dots, x_n$  are Boolean expressions.

If  $X_1$  and  $X_2$  are Boolean expressions, so are

$$(X_1), \quad X'_1, \quad X_1 + X_2, \quad X_1 \cdot X_2.$$

A **Boolean function** over  $S$  is defined as a function from  $S^n$  to  $S$  of the form

$$f(x_1, \dots, x_n) = X(x_1, \dots, x_n),$$

where  $X$  is a Boolean expression in the symbols  $x_1, \dots, x_n$  over  $S$ . A disjunctive normal form can be defined for  $f$ . Another result is that if  $X$  and  $Y$  are Boolean expressions over  $S$  and

$$X(x_1, \dots, x_n) = Y(x_1, \dots, x_n)$$

for all  $x_i \in S$ , then  $Y$  is derivable from  $X$  using the definition (Definition 11.3.1) of a Boolean algebra. Other results are that any finite Boolean algebra has  $2^n$  elements and that if two Boolean algebras both have  $2^n$  elements they are essentially the same. It follows that any finite Boolean algebra is essentially Example 11.3.3, the Boolean algebra of subsets of a finite, universal set  $U$ . The proofs of these results can be found in [Mendelson].

## Chapter 11 Review

### Section 11.1

1. Combinatorial circuit
2. Sequential circuit
3. AND gate
4. OR gate
5. NOT gate (inverter)
6. Logic table of a combinatorial circuit
7. Boolean expression
8. Literal

### Section 11.2

9. Properties of  $\wedge$ ,  $\vee$ , and  $\neg$ : associative laws; commutative laws; distributive laws; identity laws; complement laws (see Theorem 11.2.1)
10. Equal Boolean expressions
11. Equivalent combinatorial expressions
12. Combinatorial expressions are equivalent if and only if the Boolean expressions that represent them are equal.

### Section 11.3

13. Boolean algebra
14.  $x'$ : Complement of  $x$
15. Properties of Boolean algebras: Idempotent laws; bound laws; absorption laws; involution law; 0 and 1 laws; De Morgan's laws
16. Dual of statement involving Boolean expressions

17. The dual of a theorem about Boolean algebras is also a theorem.

### Section 11.4

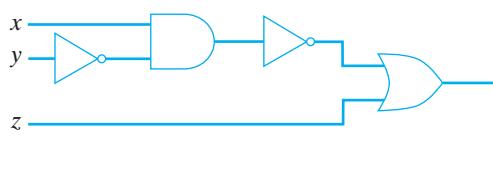
18. Exclusive-OR
19. Boolean function
20. Minterm:  $y_1 \wedge y_2 \wedge \dots \wedge y_n$ , where each  $y_i$  is  $x_i$  or  $\bar{x}_i$
21. Disjunctive normal form
22. How to write a Boolean function in disjunctive normal form (Theorem 11.4.6)
23. Maxterm:  $y_1 \vee y_2 \vee \dots \vee y_n$ , where each  $y_i$  is  $x_i$  or  $\bar{x}_i$
24. Conjunctive normal form

### Section 11.5

25. Gate
26. Functionally complete set of gates
27. The sets of gates {AND, NOT} and {OR, NOT} are functionally complete.
28. NAND gate
29. The set {NAND} is a functionally complete set of gates.
30. Minimization problem
31. Integrated circuit
32. Half-adder circuit
33. Full-adder circuit

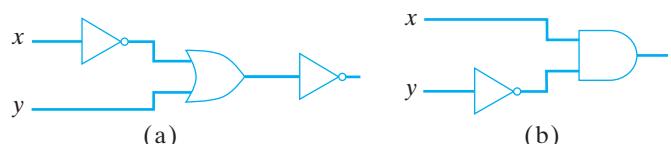
## Chapter 11 Self-Test

1. Write a Boolean expression that represents the combinatorial circuit and write the logic table.

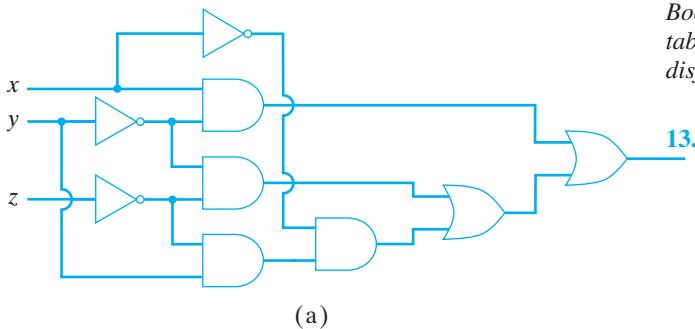


*Are the combinatorial circuits in Exercises 2 and 3 equivalent? Explain.*

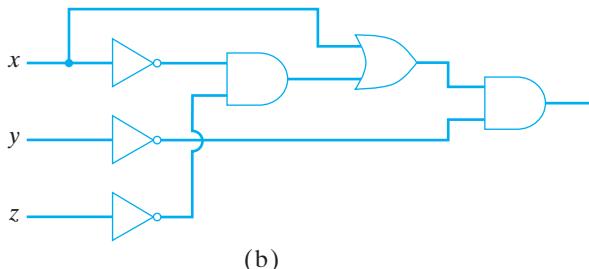
2.



3.



(a)



(b)

4. If  $U$  is a universal set and  $S = \mathcal{P}(U)$ , the power set of  $U$ , then

$$(S, \cup, \cap, \neg, \emptyset, U)$$

is a Boolean algebra. State the bound and absorption laws for this Boolean algebra.

5. Find the value of the Boolean expression

$$(x_1 \wedge x_2) \vee (\bar{x}_2 \wedge x_3)$$

if  $x_1 = x_2 = 0$  and  $x_3 = 1$ .

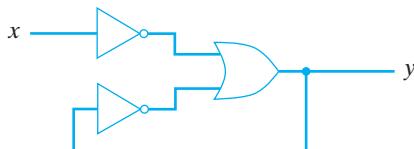
6. Find a combinatorial circuit corresponding to the Boolean expression of Exercise 5.

*Prove or disprove the equations in Exercises 7 and 8.*

7.  $(x \wedge y) \vee (\bar{x} \wedge z) \vee (\bar{x} \wedge y \wedge \bar{z}) = y \vee (\bar{x} \wedge z)$

8.  $(x \wedge y \wedge z) \vee (\bar{x} \vee \bar{z}) = (x \wedge z) \vee (\bar{x} \wedge \bar{z})$

9. Show that the following circuit is not a combinatorial circuit.



10. Prove that in any Boolean algebra,  $(x \cdot (x + y \cdot 0))' = x'$  for all  $x$  and  $y$ .

11. Write the dual of the statement of Exercise 10 and prove it.

12. Let  $U$  be the set of positive integers. Let  $S$  be the collection of finite subsets of  $U$ . Why does  $(S, \cup, \cap, \neg, \emptyset, U)$  fail to be a Boolean algebra?

In Exercises 13–16, find the disjunctive normal form of a Boolean expression having a logic table the same as the given table and draw the combinatorial circuit corresponding to the disjunctive normal form.

13.

| $x_1$ | $x_2$ | $x_3$ | $y$ |
|-------|-------|-------|-----|
| 1     | 1     | 1     | 0   |
| 1     | 1     | 0     | 0   |
| 1     | 0     | 1     | 0   |
| 1     | 0     | 0     | 1   |
| 0     | 1     | 1     | 0   |
| 0     | 1     | 0     | 0   |
| 0     | 0     | 1     | 0   |
| 0     | 0     | 0     | 0   |

14.

| $x_1$ | $x_2$ | $x_3$ | $y$ |
|-------|-------|-------|-----|
| 1     | 1     | 1     | 0   |
| 1     | 1     | 0     | 1   |
| 1     | 0     | 1     | 0   |
| 1     | 0     | 0     | 1   |
| 0     | 1     | 1     | 0   |
| 0     | 1     | 0     | 0   |
| 0     | 0     | 1     | 0   |
| 0     | 0     | 0     | 0   |

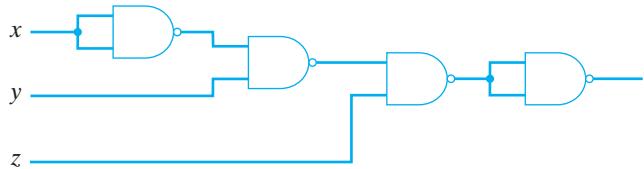
15.

| $x_1$ | $x_2$ | $x_3$ | $y$ |
|-------|-------|-------|-----|
| 1     | 1     | 1     | 1   |
| 1     | 1     | 0     | 0   |
| 1     | 0     | 1     | 0   |
| 1     | 0     | 0     | 1   |
| 0     | 1     | 1     | 0   |
| 0     | 1     | 0     | 0   |
| 0     | 0     | 1     | 0   |
| 0     | 0     | 0     | 1   |

16.

| $x_1$ | $x_2$ | $x_3$ | $y$ |
|-------|-------|-------|-----|
| 1     | 1     | 1     | 0   |
| 1     | 1     | 0     | 1   |
| 1     | 0     | 1     | 0   |
| 1     | 0     | 0     | 1   |
| 0     | 1     | 1     | 1   |
| 0     | 1     | 0     | 0   |
| 0     | 0     | 1     | 1   |
| 0     | 0     | 0     | 0   |

17. Write the logic table for the circuit



18. Find a Boolean expression in disjunctive normal form for the circuit of part (a) of Exercise 3. Use algebraic methods to simplify the disjunctive normal form. Draw the circuit corresponding to the simplified expression.

19. Design a circuit using only NAND gates to compute  $x \oplus y$ .
20. Design a full-adder circuit that uses two half adders and one OR gate.

## Chapter 11 Computer Exercises

1. Write a program that inputs a Boolean expression in  $x$  and  $y$  and prints the logic table of the expression.
2. Write a program that inputs a Boolean expression in  $x$ ,  $y$ , and  $z$  and prints the logic table of the expression.
3. Write a program that outputs the disjunctive normal form of a Boolean expression  $p(x, y)$ .
4. Write a program that outputs the conjunctive normal form of a Boolean expression  $p(x, y)$ .
5. Write a program that outputs the disjunctive normal form of a Boolean expression  $p(x, y, z)$ .
6. Write a program that outputs the conjunctive normal form of a Boolean expression  $p(x, y, z)$ .
7. Write a program that computes the two's complement of an  $n$ -bit binary number.



## Chapter 12

# AUTOMATA, GRAMMARS, AND LANGUAGES

- 12.1** Sequential Circuits and Finite-State Machines
- 12.2** Finite-State Automata
- 12.3** Languages and Grammars
- 12.4** Nondeterministic Finite-State Automata
- 12.5** Relationships Between Languages and Automata

In Chapter 11 we discussed combinatorial circuits in which the output depended only on the input. These circuits have no memory. In this chapter we begin by discussing circuits in which the output depends not only on the input but also on the state of the system at the time the input is introduced. The state of the system is determined by previous processing. In this sense, these circuits have memory. Such circuits are called *sequential circuits* and are obviously important in computer design.

Finite-state machines are abstract models of machines with a primitive internal memory. A finite-state automaton is a special kind of finite-state machine that is closely linked to a particular type of language. In the latter part of this chapter, we will discuss finite-state machines, finite-state automata, and languages in some detail.

### 12.1

#### Sequential Circuits and Finite-State Machines

##### Go Online

For more on sequential circuits and finite-state machines, see  
[goo.gl/VVncUc](http://goo.gl/VVncUc)

Operations within a digital computer are carried out at discrete intervals of time. Output depends on the state of the system as well as on the input. We will assume that the state of the system changes only at time  $t = 0, 1, \dots$ . A simple way to introduce sequencing in circuits is to introduce a **unit time delay**.

**Definition 12.1.1** ► A *unit time delay* accepts as input a bit  $x_t$  at time  $t$  and outputs  $x_{t-1}$ , the bit received as input at time  $t - 1$ . The unit time delay is drawn as shown in Figure 12.1.1.



Figure 12.1.1 Unit time delay.

As an example of the use of the unit time delay, we discuss the **serial adder**.

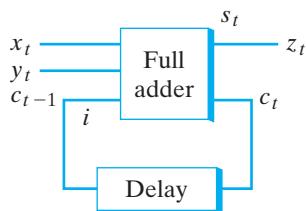
**Definition 12.1.2** ► A *serial adder* accepts as input two binary numbers

$$x = 0x_Nx_{N-1}\dots x_0 \quad \text{and} \quad y = 0y_Ny_{N-1}\dots y_0$$

and outputs the sum  $z_{N+1}z_N\dots z_0$  of  $x$  and  $y$ . The numbers  $x$  and  $y$  are input sequentially in pairs,  $x_0, y_0; \dots; x_N, y_N; 0, 0$ . The sum is output  $z_0, z_1, \dots, z_{N+1}$ .

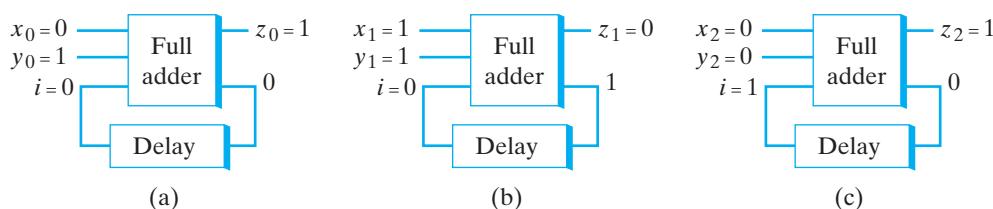
**Example 12.1.3**

**Serial-Adder Circuit** A circuit, using a unit time delay, that implements a serial adder is shown in Figure 12.1.2. Show how the serial adder computes the sum of  $x = 010$  and  $y = 011$ .



**Figure 12.1.2** A serial-adder circuit.

**SOLUTION** We begin by setting  $x_0 = 0$  and  $y_0 = 1$ . (We assume that at this instant  $i = 0$ . This can be arranged by first setting  $x = y = 0$ .) The state of the system is shown in Figure 12.1.3(a). Next, we set  $x_1 = y_1 = 1$ . The unit time delay sends  $i = 0$  as the third bit to the full adder. The state of the system is shown in Figure 12.1.3(b). Finally, we set  $x_2 = y_2 = 0$ . This time the unit time delay sends  $i = 1$  as the third bit to the full adder. The state of the system is shown in Figure 12.1.3(c). We obtain the sum  $z = 101$ .



**Figure 12.1.3** Computing  $010 + 011$  with the serial-adder circuit.

A **finite-state machine** is an abstract model of a machine with a primitive internal memory.

**Definition 12.1.4** ► A *finite-state machine M* consists of

- (a) A finite set  $\mathcal{I}$  of *input symbols*.
- (b) A finite set  $\mathcal{O}$  of *output symbols*.
- (c) A finite set  $\mathcal{S}$  of *states*.
- (d) A *next-state function*  $f$  from  $\mathcal{S} \times \mathcal{I}$  into  $\mathcal{S}$ .
- (e) An *output function*  $g$  from  $\mathcal{S} \times \mathcal{I}$  into  $\mathcal{O}$ .
- (f) An *initial state*  $\sigma \in \mathcal{S}$ .

We write  $M = (\mathcal{I}, \mathcal{O}, \mathcal{S}, f, g, \sigma)$ .

**Example 12.1.5**

Let  $\mathcal{I} = \{a, b\}$ ,  $\mathcal{O} = \{0, 1\}$ , and  $\mathcal{S} = \{\sigma_0, \sigma_1\}$ . Define the pair of functions  $f: \mathcal{S} \times \mathcal{I} \rightarrow \mathcal{S}$  and  $g: \mathcal{S} \times \mathcal{I} \rightarrow \mathcal{O}$  by the rules given in Table 12.1.1.

**TABLE 12.1.1**

|               |               | $f$        |            | $g$ |     |
|---------------|---------------|------------|------------|-----|-----|
|               |               | $a$        | $b$        | $a$ | $b$ |
| $\mathcal{S}$ | $\mathcal{I}$ |            |            |     |     |
|               | $\sigma_0$    | $\sigma_0$ | $\sigma_1$ | 0   | 1   |
|               | $\sigma_1$    | $\sigma_1$ | $\sigma_1$ | 1   | 0   |

Then  $M = (\mathcal{I}, \mathcal{O}, \mathcal{S}, f, g, \sigma_0)$  is a finite-state machine.

Table 12.1.1 is interpreted to mean

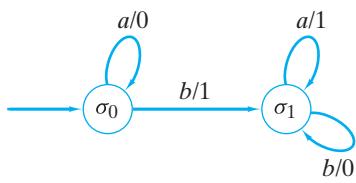
$$\begin{array}{ll} f(\sigma_0, a) = \sigma_0 & g(\sigma_0, a) = 0, \\ f(\sigma_0, b) = \sigma_1 & g(\sigma_0, b) = 1, \\ f(\sigma_1, a) = \sigma_1 & g(\sigma_1, a) = 1, \\ f(\sigma_1, b) = \sigma_0 & g(\sigma_1, b) = 0. \end{array}$$



The next-state and output functions can also be defined by a **transition diagram**. Before formally defining a transition diagram, we will illustrate how a transition diagram is constructed.

### Example 12.1.6

Draw the transition diagram for the finite-state machine of Example 12.1.5.



**Figure 12.1.4** A transition diagram.

**SOLUTION** The transition diagram is a digraph. The vertices are the states (see Figure 12.1.4). The initial state is indicated by an arrow as shown. If we are in state  $\sigma$  and inputting  $i$  causes output  $o$  and moves us to state  $\sigma'$ , we draw a directed edge from vertex  $\sigma$  to vertex  $\sigma'$  and label it  $i/o$ . For example, if we are in state  $\sigma_0$ , and we input  $a$ , Table 12.1.1 tells us that we output 0 and remain in state  $\sigma_0$ . Thus we draw a directed loop on vertex  $\sigma_0$  and label it  $a/0$  (see Figure 12.1.4). On the other hand, if we are in state  $\sigma_0$  and we input  $b$ , we output 1 and move to state  $\sigma_1$ . Thus we draw a directed edge from  $\sigma_0$  to  $\sigma_1$  and label it  $b/1$ . By considering all such possibilities, we obtain the transition diagram of Figure 12.1.4.



**Definition 12.1.7** ► Let  $M = (\mathcal{I}, \mathcal{O}, \mathcal{S}, f, g, \sigma)$  be a finite-state machine. The *transition diagram* of  $M$  is a digraph  $G$  whose vertices are the members of  $\mathcal{S}$ . An arrow designates the initial state  $\sigma$ . A directed edge  $(\sigma_1, \sigma_2)$  exists in  $G$  if there exists an input  $i$  with  $f(\sigma_1, i) = \sigma_2$ . In this case, if  $g(\sigma_1, i) = o$ , the edge  $(\sigma_1, \sigma_2)$  is labeled  $i/o$ .



We can regard the finite-state machine  $M = (\mathcal{I}, \mathcal{O}, \mathcal{S}, f, g, \sigma)$  as a simple computer. We begin in state  $\sigma$ , input a string over  $\mathcal{I}$ , and produce a string of output.

**Definition 12.1.8** ► Let  $M = (\mathcal{I}, \mathcal{O}, \mathcal{S}, f, g, \sigma)$  be a finite-state machine. An *input string* for  $M$  is a string over  $\mathcal{I}$ . The string

$$y_1 \cdots y_n$$

is the *output string* for  $M$  corresponding to the input string

$$\alpha = x_1 \cdots x_n$$

if there exist states  $\sigma_0, \dots, \sigma_n \in \mathcal{S}$  with

$$\begin{aligned} \sigma_0 &= \sigma \\ \sigma_i &= f(\sigma_{i-1}, x_i) \quad \text{for } i = 1, \dots, n; \\ y_i &= g(\sigma_{i-1}, x_i) \quad \text{for } i = 1, \dots, n. \end{aligned}$$



### Example 12.1.9

Find the output string corresponding to the input string

$$aababba \tag{12.1.1}$$

for the finite-state machine of Example 12.1.5.

**SOLUTION** Initially, we are in state  $\sigma_0$ . The first symbol input is  $a$ . We locate the outgoing edge in the transition diagram of  $M$  (Figure 12.1.4) from  $\sigma_0$  labeled  $a/x$ , which

tells us that if  $a$  is input,  $x$  is output. In our case, 0 is output. The edge points to the next state,  $\sigma_0$ . Next,  $a$  is input again. As before, we output 0 and remain in state  $\sigma_0$ . Next,  $b$  is input. In this case, we output 1 and change to state  $\sigma_1$ . Continuing in this way, we find that the output string is

$$0011001. \quad (12.1.2)$$

**Example 12.1.10 A Serial-Adder Finite-State Machine** Design a finite-state machine that performs serial addition.

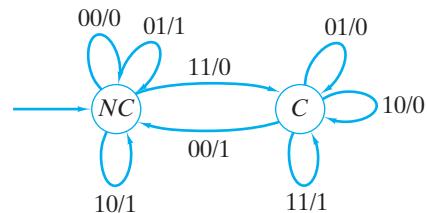
**SOLUTION** We will represent the finite-state machine by its transition diagram. Since the serial adder accepts pairs of bits, the input set is  $\{00, 01, 10, 11\}$  and the output set is  $\{0, 1\}$ .

Given an input  $xy$ , we take one of two actions: Either we add  $x$  and  $y$ , or we add  $x$ ,  $y$ , and 1, depending on whether the carry bit was 0 or 1. Thus there are two states, which we will call  $C$  (carry) and  $NC$  (no carry). The initial state is  $NC$ . At this point, we can draw the vertices and designate the initial state in our transition diagram (see Figure 12.1.5).

Next, we consider the possible inputs at each vertex. For example, if 00 is input to  $NC$ , we should output 0 and remain in state  $NC$ . Thus  $NC$  has a loop labeled 00/0. As another example, if 11 is input to  $C$ , we compute  $1 + 1 + 1 = 11$ . In this case we output 1 and remain in state  $C$ . Thus  $C$  has a loop labeled 11/1. As a final example, if we are in state  $NC$  and 11 is input, we should output 0 and move to state  $C$ . By considering all possibilities, we arrive at the transition diagram of Figure 12.1.6.



**Figure 12.1.5** Two states for the serial-adder finite-state machine.



**Figure 12.1.6** A finite-state machine that performs serial addition.

**Example 12.1.11 The SR Flip-Flop** A **flip-flop** is a basic component of digital circuits since it serves as a one-bit memory cell. The **SR flip-flop** (or **set-reset flip-flop**) can be defined by the table

| S | R | Q                                                                                                                        |
|---|---|--------------------------------------------------------------------------------------------------------------------------|
| 1 | 1 | Not allowed                                                                                                              |
| 1 | 0 | 1                                                                                                                        |
| 0 | 1 | 0                                                                                                                        |
| 0 | 0 | $\begin{cases} 1 & \text{if } S \text{ was last equal to 1} \\ 0 & \text{if } R \text{ was last equal to 1} \end{cases}$ |

The **SR** flip-flop “remembers” whether  $S$  or  $R$  was last equal to 1. (If  $Q = 1$ ,  $S$  was last equal to 1; if  $Q = 0$ ,  $R$  was last equal to 1.) We can model the **SR** flip-flop as a finite-state machine by defining two states: “ $S$  was last equal to 1” and “ $R$  was last equal to 1” (see Figure 12.1.7). We define the input to be the new values of  $S$  and  $R$ ; the notation  $sr$  means that  $S = s$  and  $R = r$ . We define  $Q$  to be the output. We have arbitrarily designated the initial state as “ $S$  was last equal to 1.” A sequential circuit implementation of the **SR** flip-flop is shown in Figure 12.1.8.

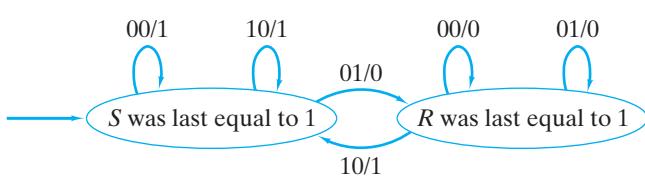


Figure 12.1.7 The SR flip-flop as a finite-state machine.

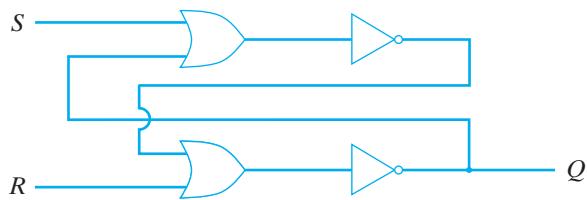


Figure 12.1.8 A sequential circuit implementation of the SR flip-flop.

## 12.1 Review Exercises

1. What is a unit time delay?
2. What is a serial adder?
3. Define *finite-state machine*.
4. What is a transition diagram?
5. What is the SR flip-flop?

## 12.1 Exercises

In Exercises 1–5, draw the transition diagram of the finite-state machine  $(\mathcal{I}, \mathcal{O}, \mathcal{S}, f, g, \sigma_0)$ .

1.  $\mathcal{I} = \{a, b\}$ ,  $\mathcal{O} = \{0, 1\}$ ,  $\mathcal{S} = \{\sigma_0, \sigma_1\}$

|               |               | f          |            | g |   |
|---------------|---------------|------------|------------|---|---|
|               |               | a          | b          | a | b |
| $\mathcal{S}$ | $\mathcal{I}$ |            |            |   |   |
|               | $\sigma_0$    | $\sigma_1$ | $\sigma_1$ | 1 | 1 |
| $\sigma_1$    | $\sigma_0$    | $\sigma_1$ |            | 0 | 1 |

2.  $\mathcal{I} = \{a, b\}$ ,  $\mathcal{O} = \{0, 1\}$ ,  $\mathcal{S} = \{\sigma_0, \sigma_1\}$

|               |               | f          |            | g |   |
|---------------|---------------|------------|------------|---|---|
|               |               | a          | b          | a | b |
| $\mathcal{S}$ | $\mathcal{I}$ |            |            |   |   |
|               | $\sigma_0$    | $\sigma_1$ | $\sigma_0$ | 0 | 0 |
| $\sigma_1$    | $\sigma_0$    | $\sigma_0$ |            | 1 | 1 |

3.  $\mathcal{I} = \{a, b\}$ ,  $\mathcal{O} = \{0, 1\}$ ,  $\mathcal{S} = \{\sigma_0, \sigma_1, \sigma_2\}$

|               |               | f          |            | g |   |
|---------------|---------------|------------|------------|---|---|
|               |               | a          | b          | a | b |
| $\mathcal{S}$ | $\mathcal{I}$ |            |            |   |   |
|               | $\sigma_0$    | $\sigma_1$ | $\sigma_1$ | 0 | 1 |
| $\sigma_1$    | $\sigma_2$    | $\sigma_1$ |            | 1 | 1 |
| $\sigma_2$    | $\sigma_0$    | $\sigma_0$ |            | 0 | 0 |

4.  $\mathcal{I} = \{a, b, c\}$ ,  $\mathcal{O} = \{0, 1\}$ ,  $\mathcal{S} = \{\sigma_0, \sigma_1, \sigma_2\}$

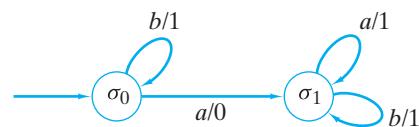
|               |               | f          |            |            | g |   |   |
|---------------|---------------|------------|------------|------------|---|---|---|
|               |               | a          | b          | c          | a | b | c |
| $\mathcal{S}$ | $\mathcal{I}$ |            |            |            |   |   |   |
|               | $\sigma_0$    | $\sigma_0$ | $\sigma_1$ | $\sigma_2$ | 0 | 1 | 0 |
| $\sigma_1$    | $\sigma_1$    | $\sigma_1$ | $\sigma_1$ | $\sigma_0$ | 1 | 1 | 1 |
| $\sigma_2$    | $\sigma_2$    | $\sigma_2$ | $\sigma_1$ | $\sigma_0$ | 1 | 0 | 0 |

5.  $\mathcal{I} = \{a, b, c\}$ ,  $\mathcal{O} = \{0, 1, 2\}$ ,  $\mathcal{S} = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$

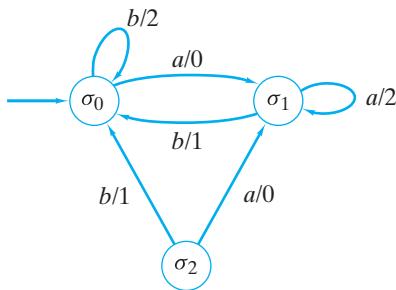
|               |               | f          |            |            | g |   |   |
|---------------|---------------|------------|------------|------------|---|---|---|
|               |               | a          | b          | c          | a | b | c |
| $\mathcal{S}$ | $\mathcal{I}$ |            |            |            |   |   |   |
|               | $\sigma_0$    | $\sigma_1$ | $\sigma_0$ | $\sigma_2$ | 1 | 1 | 2 |
| $\sigma_1$    | $\sigma_0$    | $\sigma_0$ | $\sigma_2$ | $\sigma_2$ | 2 | 0 | 0 |
| $\sigma_2$    | $\sigma_3$    | $\sigma_3$ | $\sigma_0$ | $\sigma_0$ | 1 | 0 | 1 |
| $\sigma_3$    | $\sigma_1$    | $\sigma_1$ | $\sigma_0$ | $\sigma_0$ | 2 | 0 | 2 |

In Exercises 6–10, find the sets  $\mathcal{I}$ ,  $\mathcal{O}$ , and  $\mathcal{S}$ , the initial state, and the table defining the next-state and output functions for each finite-state machine.

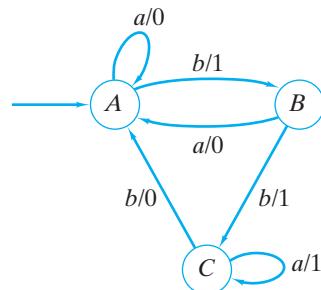
6.



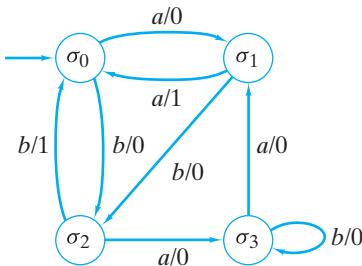
7.



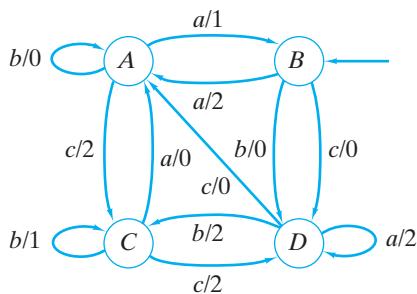
8.



9.



10.



In Exercises 11–20, find the output string for the given input string and finite-state machine.

11. abba; Exercise 1
12. abba; Exercise 2
13. aabbaba; Exercise 3
14. aabbcc; Exercise 4
15. aabaab; Exercise 5
16. aaa; Exercise 6
17. baaba; Exercise 7

18. aabbabaab; Exercise 8

19. bbababbbabaaa; Exercise 9

20. cacbccbaabac; Exercise 10

In Exercises 21–26, design a finite-state machine having the given properties. The input is always a bit string.

21. Outputs 1 if an even number of 1's have been input; otherwise, outputs 0

22. Outputs 1 if two or more 1's are input; otherwise, outputs 0

23. Outputs 1 if  $k$  1's have been input, where  $k$  is a multiple of 3; otherwise, outputs 0

24. Outputs 1 whenever it sees 101; otherwise, outputs 0

25. Outputs 1 when it sees the first 0 and until it sees another 0; thereafter, outputs 0; in all other cases, outputs 0

26. Outputs 1 when it sees 101 and thereafter; otherwise, outputs 0

27. Let  $\alpha = x_1 \cdots x_n$  be a bit string. Let  $\beta = y_1 \cdots y_n$ , where

$$y_i = \begin{cases} a & \text{if } x_i = 0 \\ b & \text{if } x_i = 1 \end{cases}$$

for  $i = 1, \dots, n$ . Let  $\gamma = y_n \cdots y_1$ .

Show that if  $\gamma$  is input to the finite-state machine of Figure 12.1.4, the output is the 2's complement of  $\alpha$  (see Algorithm 11.5.16 for a description of 2's complement).

\*28. Show that there is no finite-state machine that receives a bit string and outputs 1 whenever the number of 1's input equals the number of 0's input and outputs 0 otherwise.

\*29. Show that there is no finite-state machine that performs serial multiplication. Specifically, show that there is no finite-state machine that inputs binary numbers  $X = x_1 \cdots x_n$ ,  $Y = y_1 \cdots y_n$ , as the sequence of two-bit numbers

$$x_n y_n, \quad x_{n-1} y_{n-1}, \quad \dots, \quad x_1 y_1, \quad 00, \quad \dots, \quad 00,$$

where there are  $n$  00's, and outputs  $z_{2n}, \dots, z_1$ , where  $Z = z_1 \cdots z_{2n} = XY$ .

*Example:* If there is such a machine, to multiply  $101 \times 1001$  we would input 11,00,10,01,00,00,00,00. The first pair 11 is the pair of rightmost bits (101, 1001); the second pair 00 is the next pair of bits (101, 1001); and so on. We pad the input string with four pairs of 00's—the length of the longest number 1001 to be multiplied. Since  $101 \times 1001 = 101101$ , it is alleged that we obtain the output shown in the following table.

| Input | Output |
|-------|--------|
| 11    | 1      |
| 00    | 0      |
| 10    | 1      |
| 01    | 1      |
| 00    | 0      |
| 00    | 1      |
| 00    | 0      |
| 00    | 0      |

## 12.2 Finite-State Automata

### Go Online

A finite-state automaton simulator is at [goo.gl/VVncUc](http://goo.gl/VVncUc)

A **finite-state automaton** is a special kind of finite-state machine. Finite-state automata are of special interest because of their relationship to languages, as we shall see in Section 12.5.

**Definition 12.2.1** ► A *finite-state automaton*  $A = (\mathcal{I}, \mathcal{O}, \mathcal{S}, f, g, \sigma)$  is a finite-state machine in which the set of output symbols is  $\{0, 1\}$  and where the current state determines the last output. Those states for which the last output was 1 are called *accepting states*. ▲

### Example 12.2.2

Draw the transition diagram of the finite-state machine  $A$  defined by the table. The initial state is  $\sigma_0$ . Show that  $A$  is a finite-state automaton, and determine the set of accepting states.

|            |          | <i>f</i>   |            | <i>g</i> |          |
|------------|----------|------------|------------|----------|----------|
|            |          | <i>a</i>   | <i>b</i>   | <i>a</i> | <i>b</i> |
| <i>S</i>   | <i>I</i> |            |            |          |          |
|            |          | $\sigma_1$ | $\sigma_0$ | 1        | 0        |
| $\sigma_0$ |          | $\sigma_2$ | $\sigma_0$ | 1        | 0        |
| $\sigma_1$ |          | $\sigma_2$ | $\sigma_0$ | 1        | 0        |
| $\sigma_2$ |          |            |            |          |          |

**SOLUTION** The transition diagram is shown in Figure 12.2.1. If we are in state  $\sigma_0$ , the last output was 0. If we are in either state  $\sigma_1$  or  $\sigma_2$ , the last output was 1; thus  $A$  is a finite-state automaton. The accepting states are  $\sigma_1$  and  $\sigma_2$ .

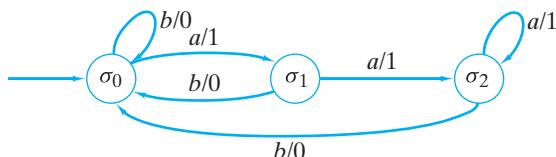


Figure 12.2.1 The transition diagram for Example 12.2.2. ▲

Example 12.2.2 shows that the finite-state machine defined by a transition diagram will be a finite-state automaton if the set of output symbols is  $\{0, 1\}$  and if, for each state  $\sigma$ , all incoming edges to  $\sigma$  have the same output label.

The transition diagram of a finite-state automaton is usually drawn with the accepting states in double circles and the output symbols omitted. When the transition diagram of Figure 12.2.1 is redrawn in this way, we obtain the transition diagram of Figure 12.2.2.

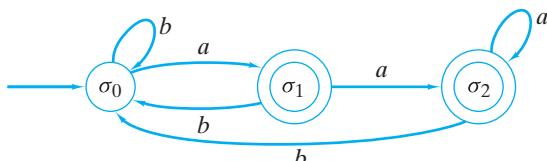
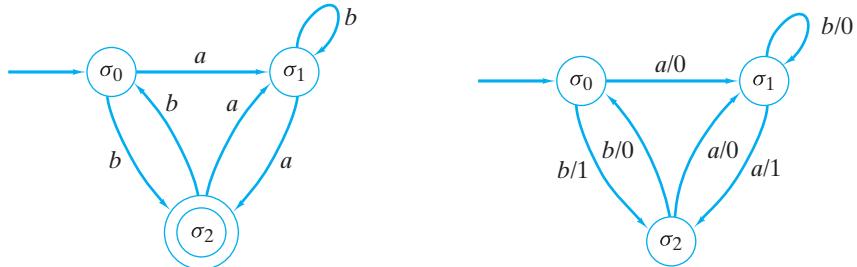


Figure 12.2.2 The transition diagram of Figure 12.2.1 redrawn with accepting states in double circles and output symbols omitted. ▲

**Example 12.2.3**

Draw the transition diagram of the finite-state automaton of Figure 12.2.3 as a transition diagram of a finite-state machine.

**SOLUTION** Since  $\sigma_2$  is an accepting state, we label all its incoming edges with output 1 (see Figure 12.2.4). The states  $\sigma_0$  and  $\sigma_1$  are not accepting, so we label all their incoming edges with output 0. We obtain the transition diagram of Figure 12.2.4.



**Figure 12.2.3** A finite-state automaton.

**Figure 12.2.4** The finite-state automaton of Figure 12.2.3 redrawn as a transition diagram of a finite-state machine. ▲

As an alternative to Definition 12.2.1, we can regard a finite-state automaton  $A$  as consisting of

1. A finite set  $\mathcal{I}$  of *input symbols*
2. A finite set  $\mathcal{S}$  of *states*
3. A *next-state function*  $f$  from  $\mathcal{S} \times \mathcal{I}$  into  $\mathcal{S}$
4. A subset  $\mathcal{A}$  of  $\mathcal{S}$  of *accepting states*
5. An *initial state*  $\sigma \in \mathcal{S}$ .

If we use this characterization, we write  $A = (\mathcal{I}, \mathcal{S}, f, \mathcal{A}, \sigma)$ .

**Example 12.2.4**

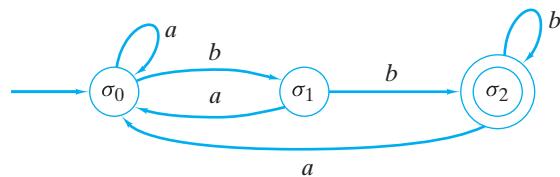
The transition diagram of the finite-state automaton  $A = (\mathcal{I}, \mathcal{S}, f, \mathcal{A}, \sigma)$ , where

$$\mathcal{I} = \{a, b\}, \quad \mathcal{S} = \{\sigma_0, \sigma_1, \sigma_2\}, \quad \mathcal{A} = \{\sigma_2\}, \quad \sigma = \sigma_0,$$

and  $f$  is given by the following table

|               |            | $f$        |            |
|---------------|------------|------------|------------|
|               |            | $a$        | $b$        |
| $\mathcal{I}$ | $\sigma_0$ | $\sigma_0$ | $\sigma_1$ |
|               | $\sigma_1$ | $\sigma_0$ | $\sigma_2$ |
| $\sigma_2$    | $\sigma_0$ | $\sigma_2$ |            |

is shown in Figure 12.2.5.



**Figure 12.2.5** The transition diagram for Example 12.2.4. ▲

If a string is input to a finite-state automaton, we will end at either an accepting or a nonaccepting state. The status of this final state determines whether the string is **accepted** by the finite-state automaton.

**Definition 12.2.5** ► Let  $A = (\mathcal{I}, \mathcal{S}, f, \mathcal{A}, \sigma)$  be a finite-state automaton. Let  $\alpha = x_1 \cdots x_n$  be a string over  $\mathcal{I}$ . If there exist states  $\sigma_0, \dots, \sigma_n$  satisfying

- (a)  $\sigma_0 = \sigma$
- (b)  $f(\sigma_{i-1}, x_i) = \sigma_i$  for  $i = 1, \dots, n$
- (c)  $\sigma_n \in \mathcal{A}$ ,

we say that  $\alpha$  is *accepted* by  $A$ . The null string is accepted if and only if  $\sigma \in \mathcal{A}$ . We let  $\text{Ac}(A)$  denote the set of strings accepted by  $A$  and we say that  $A$  *accepts*  $\text{Ac}(A)$ .

Let  $\alpha = x_1 \cdots x_n$  be a string over  $\mathcal{I}$ . Define states  $\sigma_0, \dots, \sigma_n$  by conditions (a) and (b) above. We call the (directed) path  $(\sigma_0, \dots, \sigma_n)$  the path *representing*  $\alpha$  in  $A$ . ◀

It follows from Definition 12.2.5 that if the path  $P$  represents the string  $\alpha$  in a finite-state automaton  $A$ , then  $A$  accepts  $\alpha$  if and only if  $P$  ends at an accepting state.

### Example 12.2.6

Is the string  $abaa$  accepted by the finite-state automaton of Figure 12.2.2?

**SOLUTION** We begin at state  $\sigma_0$ . When  $a$  is input, we move to state  $\sigma_1$ . When  $b$  is input, we move to state  $\sigma_0$ . When  $a$  is input, we move to state  $\sigma_1$ . Finally, when the last symbol  $a$  is input, we move to state  $\sigma_2$ . The path  $(\sigma_0, \sigma_1, \sigma_0, \sigma_1, \sigma_2)$  represents the string  $abaa$ . Since the final state  $\sigma_2$  is an accepting state, the string  $abaa$  is accepted by the finite-state automaton of Figure 12.2.2. ◀

### Example 12.2.7

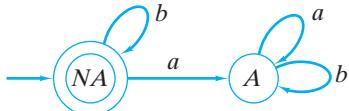
Is the string  $\alpha = abbabba$  accepted by the finite-state automaton of Figure 12.2.3?

**SOLUTION** The path representing  $\alpha$  terminates at  $\sigma_1$ . Since  $\sigma_1$  is not an accepting state, the string  $\alpha$  is not accepted by the finite-state automaton of Figure 12.2.3. ◀

We next give two examples illustrating design problems.

### Example 12.2.8

Design a finite-state automaton that accepts precisely those strings over  $\{a, b\}$  that contain no  $a$ 's.



**Figure 12.2.6** A finite-state automaton that accepts precisely those strings over  $\{a, b\}$  that contain no  $a$ 's.

**SOLUTION** The idea is to use two states:

A: An  $a$  was found.

NA: No  $a$ 's were found.

The state  $NA$  is the initial state and the only accepting state. It is now a simple matter to draw the edges (see Figure 12.2.6). Notice that the finite-state automaton correctly accepts the null string. ◀

### Example 12.2.9

Design a finite-state automaton that accepts precisely those strings over  $\{a, b\}$  that contain an odd number of  $a$ 's.

**SOLUTION** This time the two states are

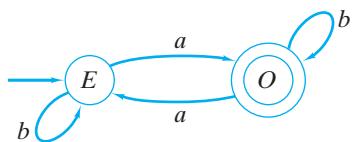
E: An even number of  $a$ 's was found.

O: An odd number of  $a$ 's was found.

The initial state is  $E$  and the accepting state is  $O$ . We obtain the transition diagram shown in Figure 12.2.7. 

A finite-state automaton is essentially an algorithm to decide whether or not a given string is accepted. As an example, we convert the transition diagram of Figure 12.2.7 to an algorithm.

### Algorithm 12.2.10



**Figure 12.2.7** A finite-state automaton that accepts precisely those strings over  $\{a, b\}$  that contain an odd number of  $a$ 's.

This algorithm determines whether a string over  $\{a, b\}$  is accepted by the finite-state automaton whose transition diagram is given in Figure 12.2.7.

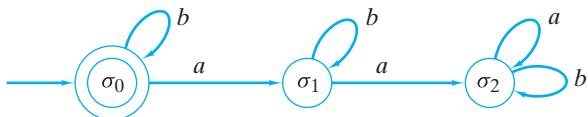
**Input:**  $n$ , the length of the string ( $n = 0$  designates the null string);  $s_1s_2 \dots s_n$ , the string  
**Output:** “Accept” if the string is accepted  
 “Reject” if the string is not accepted

```
fsa(s, n) {
 state = 'E'
 for i = 1 to n {
 if (state == 'E' & s_i == 'a')
 state = 'O'
 if (state == 'O' & s_i == 'a')
 state = 'E'
 }
 if (state == 'O')
 return "Accept"
 else
 return "Reject"
}
```

If two finite-state automata accept precisely the same strings, we say that the automata are **equivalent**.

**Definition 12.2.11** ▶ The finite-state automata  $A$  and  $A'$  are *equivalent* if  $\text{Ac}(A) = \text{Ac}(A')$ . 

**Example 12.2.12** It can be verified that the finite-state automata of Figures 12.2.6 and 12.2.8 are equivalent (see Exercise 33).



**Figure 12.2.8** A finite-state automaton equivalent to that in Figure 12.2.6. 

If we define a relation  $R$  on a set of finite-state automata by the rule  $A R A'$  if  $A$  and  $A'$  are equivalent (in the sense of Definition 12.2.11),  $R$  is an equivalence relation. Each equivalence class consists of a set of mutually equivalent finite-state automata.

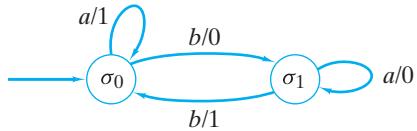
## 12.2 Review Exercises

- Define finite-state automaton.
- What does it mean for a string to be accepted by a finite-state automaton?
- What are equivalent finite-state automata?

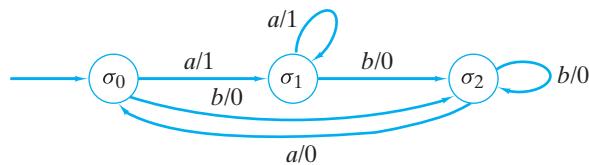
## 12.2 Exercises

In Exercises 1–3, show that each finite-state machine is a finite-state automaton and redraw the transition diagram as the diagram of a finite-state automaton.

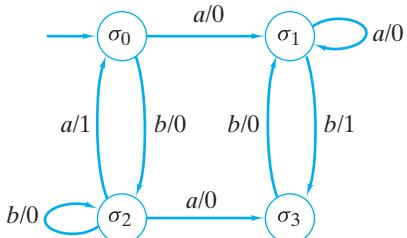
1.



2.

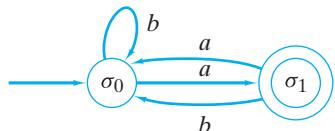


3.

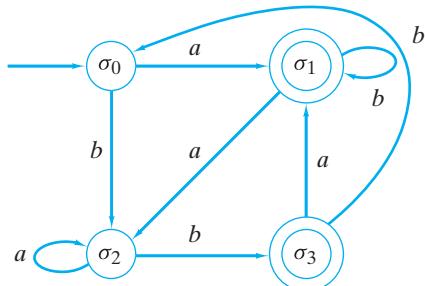


In Exercises 4–6, redraw the transition diagram of the finite-state automaton as the transition diagram of a finite-state machine.

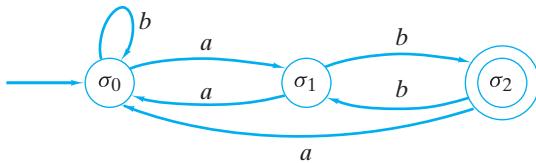
4.



5.



6.



In Exercises 7–9, draw the transition diagram of the finite-state automaton  $(\mathcal{I}, \mathcal{S}, f, \mathcal{A}, \sigma_0)$ .

- $\mathcal{I} = \{a, b\}$ ,  $\mathcal{S} = \{\sigma_0, \sigma_1, \sigma_2\}$ ,  $\mathcal{A} = \{\sigma_0\}$

|               | $f$        |            |
|---------------|------------|------------|
| $\mathcal{I}$ | $a$        | $b$        |
| $\mathcal{S}$ |            |            |
| $\sigma_0$    | $\sigma_1$ | $\sigma_0$ |
| $\sigma_1$    | $\sigma_2$ | $\sigma_0$ |
| $\sigma_2$    | $\sigma_0$ | $\sigma_2$ |

- $\mathcal{I} = \{a, b, c\}$ ,  $\mathcal{S} = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$ ,  $\mathcal{A} = \{\sigma_0, \sigma_2\}$

|               | $f$        |            |            |
|---------------|------------|------------|------------|
| $\mathcal{I}$ | $a$        | $b$        | $c$        |
| $\mathcal{S}$ |            |            |            |
| $\sigma_0$    | $\sigma_1$ | $\sigma_0$ | $\sigma_2$ |
| $\sigma_1$    | $\sigma_0$ | $\sigma_3$ | $\sigma_0$ |
| $\sigma_2$    | $\sigma_3$ | $\sigma_2$ | $\sigma_0$ |
| $\sigma_3$    | $\sigma_1$ | $\sigma_0$ | $\sigma_1$ |

- $\mathcal{I} = \{a, b\}$ ,  $\mathcal{S} = \{\sigma_0, \sigma_1, \sigma_2\}$ ,  $\mathcal{A} = \{\sigma_0, \sigma_2\}$

|               | $f$        |            |
|---------------|------------|------------|
| $\mathcal{I}$ | $a$        | $b$        |
| $\mathcal{S}$ |            |            |
| $\sigma_0$    | $\sigma_1$ | $\sigma_1$ |
| $\sigma_1$    | $\sigma_0$ | $\sigma_2$ |
| $\sigma_2$    | $\sigma_0$ | $\sigma_1$ |

10. For each finite-state automaton in Exercises 1–6, find the sets  $\mathcal{I}$ ,  $\mathcal{S}$ , and  $\mathcal{A}$ , the initial state, and the table defining the next-state function.
11. Which of the finite-state machines of Exercises 1–10, Section 12.1, are finite-state automata?
12. What must the table of a finite-state machine  $M$  look like in order for  $M$  to be a finite-state automaton?

*In Exercises 13–17, determine whether the given string is accepted by the given finite-state automaton.*

13.  $abbaa$ ; Figure 12.2.2      14.  $abbaa$ ; Figure 12.2.3

15.  $aabaabb$ ; Figure 12.2.5      16.  $aaababbab$ ; Exercise 5

17.  $aaabbaaab$ ; Exercise 6

18. Show that a string  $\alpha$  over  $\{a, b\}$  is accepted by the finite-state automaton of Figure 12.2.2 if and only if  $\alpha$  ends with  $a$ .

19. Show that a string  $\alpha$  over  $\{a, b\}$  is accepted by the finite-state automaton of Figure 12.2.5 if and only if  $\alpha$  ends with  $bb$ .

★20. Characterize the strings accepted by the finite-state automata of Exercises 1–9.

*In Exercises 21–31, draw the transition diagram of a finite-state automaton that accepts the given set of strings over  $\{a, b\}$ .*

21. Even number of  $a$ 's      22. At least one  $b$

23. Exactly one  $b$       24. Exactly two  $a$ 's

25. Contains  $m$   $a$ 's, where  $m$  is a multiple of 3

26. At least two  $a$ 's

27. Starts with  $baa$       28. Contains  $abba$

29. Every  $b$  is followed by  $a$       30. Ends with  $aba$

★31. Starts with  $ab$  and ends with  $baa$

32. Write algorithms, similar to Algorithm 12.2.10, that decide whether or not a given string is accepted by the finite-state automata of Exercises 1–9.

33. Give a formal argument to show that the finite-state automata of Figures 12.2.6 and 12.2.8 are equivalent.

34. Let  $L$  be a finite set of strings over  $\{a, b\}$ . Show that there is a finite-state automaton that accepts  $L$ .

35. Let  $L$  be the set of strings accepted by the finite-state automaton of Exercise 5. Let  $S$  denote the set of all strings over  $\{a, b\}$ . Design a finite-state automaton that accepts  $S - L$ .

36. Let  $L_i$  be the set of strings accepted by the finite-state automaton  $A_i = (\mathcal{I}, \mathcal{S}_i, f_i, \mathcal{A}_i, \sigma_i)$ ,  $i = 1, 2$ . Let

$$A = (\mathcal{I}, \mathcal{S}_1 \times \mathcal{S}_2, f, \mathcal{A}, \sigma),$$

where

$$f((S_1, S_2), x) = (f_1(S_1, x), f_2(S_2, x))$$

$$\mathcal{A} = \{(A_1, A_2) \mid A_1 \in \mathcal{A}_1 \text{ and } A_2 \in \mathcal{A}_2\}$$

$$\sigma = (\sigma_1, \sigma_2).$$

Show that  $\text{Ac}(A) = L_1 \cap L_2$ .

37. Let  $L_i$  be the set of strings accepted by the finite-state automaton  $A_i = (\mathcal{I}, \mathcal{S}_i, f_i, \mathcal{A}_i, \sigma_i)$ ,  $i = 1, 2$ . Let

$$A = (\mathcal{I}, \mathcal{S}_1 \times \mathcal{S}_2, f, \mathcal{A}, \sigma),$$

where

$$f((S_1, S_2), x) = (f_1(S_1, x), f_2(S_2, x))$$

$$\mathcal{A} = \{(A_1, A_2) \mid A_1 \in \mathcal{A}_1 \text{ or } A_2 \in \mathcal{A}_2\}$$

$$\sigma = (\sigma_1, \sigma_2).$$

Show that  $\text{Ac}(A) = L_1 \cup L_2$ .

*In Exercises 38–42, let  $L_i = \text{Ac}(A_i)$ ,  $i = 1, 2$ . Draw the transition diagrams of the finite-state automata that accept  $L_1 \cap L_2$  and  $L_1 \cup L_2$ .*

38.  $A_1$  given by Exercise 4;  $A_2$  given by Exercise 6

39.  $A_1$  given by Exercise 4;  $A_2$  given by Exercise 5

40.  $A_1$  given by Exercise 6;  $A_2$  given by Exercise 5

41.  $A_1$  given by Exercise 5;  $A_2$  given by Exercise 5

42.  $A_1$  given by Figure 12.5.7, Section 12.5;  $A_2$  given by Exercise 5

## 12.3 Languages and Grammars

According to *Merriam-Webster's Collegiate® Dictionary*, language is “the words, their pronunciation, and the methods of combining them used and understood by a community.”<sup>†</sup> Such languages are often called **natural languages** to distinguish them from **formal languages**, which are used to model natural languages and to communicate with computers. The rules of a natural language are very complex and difficult to characterize completely. On the other hand, it is possible to specify completely the rules by which certain formal languages are constructed. We begin with the definition of a formal language.

<sup>†</sup>By Permission. From *Merriam-Webster's Collegiate® Dictionary*, 11th Edition ©2016 Merriam Webster, Inc. ([www.Merriam-Webster.com](http://www.Merriam-Webster.com)).

**Definition 12.3.1** ► Let  $A$  be a finite set. A (*formal*) *language*  $L$  over  $A$  is a subset of  $A^*$ , the set of all strings over  $A$ .

**Example 12.3.2**

Let  $A = \{a, b\}$ . The set  $L$  of all strings over  $A$  containing an odd number of  $a$ 's is a language over  $A$ . As we saw in Example 12.2.9,  $L$  is precisely the set of strings over  $A$  accepted by the finite-state automaton of Figure 12.2.7.

One way to define a language is to give a list of rules that the language is assumed to obey.

**Definition 12.3.3** ► A *phrase-structure grammar* (or, simply, *grammar*)  $G$  consists of

- (a) A finite set  $N$  of *nonterminal symbols*
- (b) A finite set  $T$  of *terminal symbols* where  $N \cap T = \emptyset$
- (c) A finite subset  $P$  of  $[(N \cup T)^* - T^*] \times (N \cup T)^*$ , called the set of *productions*
- (d) A *starting symbol*  $\sigma \in N$ .

We write  $G = (N, T, P, \sigma)$ .

A production  $(A, B) \in P$  is usually written

$$A \rightarrow B.$$

Definition 12.3.3(c) states that in the production  $A \rightarrow B$ ,  $A \in (N \cup T)^* - T^*$  and  $B \in (N \cup T)^*$ ; thus  $A$  must include at least one nonterminal symbol, whereas  $B$  can consist of any combination of nonterminal and terminal symbols.

**Example 12.3.4**

Let

$$N = \{\sigma, S\}$$

$$T = \{a, b\}$$

$$P = \{\sigma \rightarrow b\sigma, \sigma \rightarrow aS, S \rightarrow bS, S \rightarrow b\}.$$

Then  $G = (N, T, P, \sigma)$  is a grammar.

Given a grammar  $G$ , we can construct a language  $L(G)$  from  $G$  by using the productions to derive the strings that make up  $L(G)$ . The idea is to start with the starting symbol and then repeatedly use productions until a string of terminal symbols is obtained. The language  $L(G)$  is the set of all such strings obtained. Definition 12.3.5 gives the formal details.

**Definition 12.3.5** ► Let  $G = (N, T, P, \sigma)$  be a grammar.

If  $\alpha \rightarrow \beta$  is a production and  $x\alpha y \in (N \cup T)^*$ , we say that  $x\beta y$  is *directly derivable* from  $x\alpha y$  and write

$$x\alpha y \Rightarrow x\beta y.$$

If  $\alpha_i \in (N \cup T)^*$  for  $i = 1, \dots, n$ , and  $\alpha_{i+1}$  is directly derivable from  $\alpha_i$  for  $i = 1, \dots, n-1$ , we say that  $\alpha_n$  is *derivable from*  $\alpha_1$  and write

$$\alpha_1 \Rightarrow \alpha_n.$$

We call

$$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n$$

the *derivation of*  $\alpha_n$  (*from*  $\alpha_1$ ). By convention, any element of  $(N \cup T)^*$  is derivable from itself.

The *language generated by G*, written  $L(G)$ , consists of all strings over  $T$  derivable from  $\sigma$ .

### Example 12.3.6

Let  $G$  be the grammar of Example 12.3.4.

The string  $abSbb$  is directly derivable from  $aSbb$ , written

$$aSbb \Rightarrow abSbb,$$

by using the production  $S \rightarrow bS$ .

The string  $bbab$  is derivable from  $\sigma$ , written

$$\sigma \Rightarrow bbab.$$

The derivation is

$$\sigma \Rightarrow b\sigma \Rightarrow bb\sigma \Rightarrow bbaS \Rightarrow bbab.$$

The only derivations from  $\sigma$  are

$$\begin{aligned} \sigma &\Rightarrow b\sigma \\ &\vdots \\ &\Rightarrow b^n\sigma \quad n \geq 0 \\ &\Rightarrow b^n aS \\ &\vdots \\ &\Rightarrow b^n ab^{m-1}S \\ &\Rightarrow b^n ab^m \quad n \geq 0, \quad m \geq 1. \end{aligned}$$

Thus  $L(G)$  consists of the strings over  $\{a, b\}$  containing precisely one  $a$  that ends with  $b$ .

### Go Online

For more on Backus normal form, see  
[goo.gl/VVncUC](http://goo.gl/VVncUC)

An alternative way to state the productions of a grammar is by using **Backus normal form** (or **Backus–Naur form** or **BNF**). In BNF the nonterminal symbols typically begin with “ $<$ ” and end with “ $>$ .” The production  $S \rightarrow T$  is written  $S ::= T$ . Productions of the form

$$S ::= T_1, \quad S ::= T_2, \quad \dots, \quad S ::= T_n$$

may be combined as

$$S ::= T_1 \mid T_2 \mid \dots \mid T_n.$$

The bar “ $|$ ” is read “or.”

### Example 12.3.7

**A Grammar for Integers** An integer is defined as a string consisting of an optional sign (+ or –) followed by a string of digits (0 through 9). The following grammar generates all integers.

```

< digit > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
< integer > ::= < signed integer > | < unsigned integer >
< signed integer > ::= + < unsigned integer > | - < unsigned integer >
< unsigned integer > ::= < digit > | < digit > < unsigned integer >

```

The starting symbol is  $< \text{integer} >$ .

For example, the derivation of the integer  $-901$  is

$$\begin{aligned}
 <\text{integer}> &\Rightarrow <\text{signed integer}> \\
 &\Rightarrow - <\text{unsigned integer}> \\
 &\Rightarrow - <\text{digit}> <\text{unsigned integer}> \\
 &\Rightarrow - <\text{digit}> <\text{digit}> <\text{unsigned integer}> \\
 &\Rightarrow - <\text{digit}> <\text{digit}> <\text{digit}> \\
 &\Rightarrow -9 <\text{digit}> <\text{digit}> \\
 &\Rightarrow -90 <\text{digit}> \\
 &\Rightarrow -901.
 \end{aligned}$$

In the notation of Definition 12.3.3, this language consists of

1. The set  $N = \{<\text{digit}>, <\text{integer}>, <\text{signed integer}>, <\text{unsigned integer}>\}$  of nonterminal symbols
2. The set  $T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -\}$  of terminal symbols
3. The productions

$$\begin{aligned}
 <\text{digit}> &\rightarrow 0, \dots, <\text{digit}> \rightarrow 9 \\
 <\text{integer}> &\rightarrow <\text{signed integer}> \\
 <\text{integer}> &\rightarrow <\text{unsigned integer}> \\
 <\text{signed integer}> &\rightarrow + <\text{unsigned integer}> \\
 <\text{signed integer}> &\rightarrow - <\text{unsigned integer}> \\
 <\text{unsigned integer}> &\rightarrow <\text{digit}> \\
 <\text{unsigned integer}> &\rightarrow <\text{digit}> <\text{unsigned integer}>
 \end{aligned}$$

4. The starting symbol  $<\text{integer}>$ .

Computer languages, such as Java and C++, are typically specified in BNF. Example 12.3.7 shows how an integer constant in a computer language might be specified in BNF.

Grammars are classified according to the types of productions that define the grammars.

**Definition 12.3.8 ▶** Let  $G$  be a grammar and let  $\lambda$  denote the null string.

- (a) If every production is of the form

$$\alpha A \beta \rightarrow \alpha \delta \beta, \quad \text{where } \alpha, \beta \in (N \cup T)^*, \quad A \in N, \\
 \delta \in (N \cup T)^* - \{\lambda\}, \tag{12.3.1}$$

we call  $G$  a *context-sensitive* (or *type 1*) grammar.

- (b) If every production is of the form

$$A \rightarrow \delta, \quad \text{where } A \in N, \quad \delta \in (N \cup T)^*, \tag{12.3.2}$$

we call  $G$  a *context-free* (or *type 2*) grammar.

- (c) If every production is of the form

$$A \rightarrow a \text{ or } A \rightarrow aB \text{ or } A \rightarrow \lambda, \quad \text{where } A, B \in N, \quad a \in T,$$

we call  $G$  a *regular* (or *type 3*) grammar.