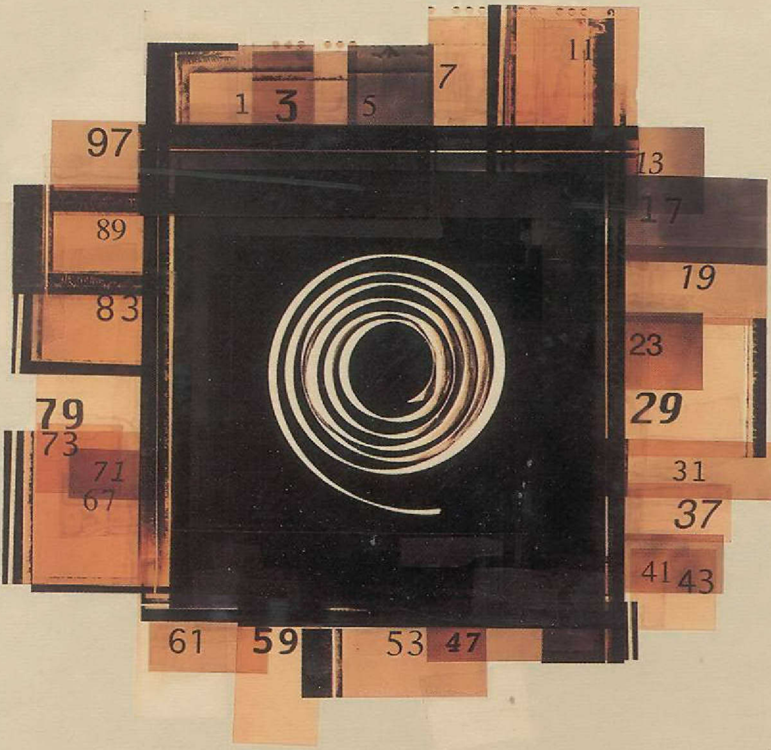
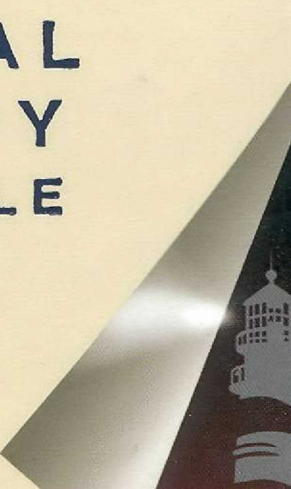


Discrete Mathematics

SECOND EDITION

James L. Hein



Discrete Mathematics

Second Edition

James L. Hein

Portland State University

JONES AND BARTLETT PUBLISHERS

Sudbury, Massachusetts

BOSTON TORONTO LONDON SINGAPORE

World Headquarters
Jones and Bartlett Publishers
40 Tall Pine Drive
Sudbury, MA 01776
978-443-5000
info@jbpub.com
www.jbpub.com

Jones and Bartlett Publishers Canada
2406 Nikanna Road
Mississauga, ON L5C 2W6
CANADA

Jones and Bartlett Publishers International
Barb House, Barb Mews
London W6 7PA
UK

Library of Congress Cataloging-in-Publication Data

Hein, James L.
Discrete mathematics / James L. Hein.–2nd ed.
p. cm.
Includes bibliographical references and index.
ISBN 0-7637-2210-3
1. Computer science–Mathematics. I. Title.
QA76.9.M35 H46 2002
004'.01'51–dc21
2002069533

Editor-in-Chief–College: J. Michael Stranz
Production Manager: Amy Rose
Senior Marketing Manager: Nathan Schultz
Associate Production Editor: Tara McCormick
Production Assistant: Karen Ferreira
Editorial Assistant: Theresa DiDonato
Composition: Northeast Compositors, Inc.
Technical Artist: George Nichols
Text Design: Anne Spencer
Cover Design: Kristin E. Ohlin
Printing and Binding: Courier Westford
Cover Printing: John P. Pow

This book was typeset in Textures 2.1 on a Macintosh G4. The font families used were Computer Modern, Rotis Sans Serif, Rotis Serif, and Prestige Elite. The first printing was printed on 40# Highland Plus.

Printed in the United States of America
06 05 04 03 02 10 9 8 7 6 5 4 3 2 1

To My Mother, Ruth Holzer Hein

She taught all eight grades in a one-room country schoolhouse in Minnesota. One cold winter morning after a snowstorm, she was riding a two-horse sleigh to school when one of the horses fell dead in its tracks. She ran to the nearest farm and called home. Her father and brother harnessed a fresh horse and met her back at the sleigh, where they harnessed the new team and drove her to school so that she could open the door and start a fire for her waiting students.

I hope that this book opens up the door and starts a fire for waiting students too.

Preface

The last thing one discovers in writing a book is what to put first.
—Blaise Pascal (1623–1662)

This book is designed for an introductory course in discrete mathematics for the prospective computer scientist, applied mathematician, or engineer who would like to learn how the ideas apply to computer science. The choice of topics—and the depth and breadth of coverage—reflects the desire to provide students with the foundations needed to successfully complete courses in undergraduate computer science programs. The book is the outgrowth of a course at Portland State University that has evolved over twenty years from a course for upper-division students into a course for sophomores.

The book can be read by anyone with a good background in high school mathematics; therefore, it could also be used at the freshman level or at the advanced high school level. Although the book is intended for future computer scientists, applied mathematicians, or engineers, it may also be suitable for a wider audience. For example, it could be used in courses for students who intend to teach discrete mathematics in high school.

This book differs in several ways from current books about discrete mathematics. It presents an elementary and unified introduction to a collection of topics that previously have not been available in a single source. A major feature of the book is the unification of the material so that it doesn't fragment into a vast collection of seemingly unrelated ideas. This is accomplished through organization and focus.

The book is organized more along the lines of technique than on a subject-by-subject basis. The focus throughout the book is on the computation and construction of objects. Therefore, many traditional topics are dispersed throughout the text to places where they fit naturally with the techniques under discussion. For example, to read about properties of—and techniques for processing—natural numbers, lists, strings, graphs, or trees, it's necessary to look in the index or scan the table of contents to find the several places where they are found.

The logic coverage is much more extensive than in current books at this level. Logic is of fundamental importance in computer science not only for its use in problem solving but also for its use in formal specification of programs, formal verification of programs, and its growing use in many areas such as databases, artificial intelligence, robotics, automatic reasoning systems, and logic programming languages.

Logic is also dispersed throughout the text. For example, we introduce informal proof techniques in the first section of Chapter 1. Then we use informal logic without much comment until Chapter 4, where inductive proof techniques are presented. After the informal use of logic is well in hand, we move to the formal aspects of logic in Chapters 6 and 7, where equivalence proofs and inference-based proofs are introduced. Formal logic is applied to proving correctness properties of programs in Chapter 8, where we also introduce higher forms of logic. We introduce automatic reasoning and logic programming in Chapter 9.

The coverage of algebraic structures differs from that in other texts. In Chapter 10, we give elementary introductions to algebras and algebraic techniques that apply directly to computer science. In addition to the traditional topic of Boolean algebra, we introduce algebras for abstract data types, relational algebra for relational databases, functional algebra for reasoning about programs, congruences with applications to cryptology, and a few other algebraic ideas that are directly applicable to computing problems.

Some Notes

- Each chapter begins with a chapter guide that gives a brief outline of the topics to be covered in each section.

- Each chapter ends with a chapter summary that gives a brief description of the main ideas covered in the chapter.

- Algorithms in the text are presented in a variety of ways. Some are simply a few sentences of explanation. Others are presented in a more traditional notation. For example, we use assignment statements like x := t and control statements like **if** A **then** B **else** C **fi**, **while** A **do** B **od**, and **for** $i :=$ 1 **to** 10 **do** C **od**. We avoid the use of begin-end or {-} pairs by using indentation. We'll also present some algorithms as logic programs after logic programming has been introduced.

- The word "proof" makes some people feel uncomfortable. It shouldn't, but it does. Maybe words like "show" or "verify" make you cringe. Most of the time we'll discuss things informally and incorporate proofs as part of the prose. At times we'll start a proof with the word "Proof," and we'll end it with QED. QED is short for the Latin phrase *quod erat demonstrandum,* which means "which was to be proved."

- A laboratory component for a course is a natural way to motivate and study the topics of the book. The course at Portland State University has evolved into a laboratory course. The ideal laboratory component uses an interactive, exploratory language such as Prolog or one of the various mathematical computing systems. The labs seem to work quite well when experiments are short and specific so that instant feedback is obtained when trying to solve a problem. Visit http://discretestructures.jbpub.com/ for a copy of both a Maple Lab Manual and a Prolog Lab Manual.

Notes for the Second Edition

- Almost every section has been rewritten to clarify and update the exposition. In addition, each section now contains many subtopic headings to identify specific areas of discussion.

- Several hundred new exercises have been added to the book so that there are now over 1,500 exercises. Answers are provided for about half of the exercises. These exercises are identified with colored numbers.

- The exercises at the end of each section are now listed by topic and ordered by difficulty within each topic. There is also a collection of proofs and/or challenges at the end of each set of exercises.

- The number of examples has been increased to make more connections between ideas and applications.

- Special attention has been paid to the report by the ACM/IEEE Joint Task Force on Computing Curricula entitled "Computing Curricula 2001." The book covers all topics listed in the report for the area of discrete structures (DS), which includes logic. The book also covers topics from the following areas listed in the report: recursion (PF4); basic algorithm analysis (AM); formal methods (SE 10); knowledge representation and reasoning (IS3).

Using the Book

As with most books, there are some dependencies among the topics. For example, all parts of the book depend on the introductory material contained in Chapter 1 and Section 2.1. But you should feel free to jump into the book at whatever topic suits your fancy and then refer back to unfamiliar definitions. Here are a few more topics with associated dependencies:

- Inductive Definitions: They are used throughout the text after being introduced in Section 3.1.
- Recursively Defined Functions: They are discussed in Section 3.2 and depend somewhat on inductive definitions in Section 3.1.
- Logic: Informal proof techniques are introduced in Section 1.1. The technique of proof by induction is covered in Section 4.4, which can be read independently with only a few references back to unfamiliar definitions. Chapter 6 and Chapter 7 should be read in order. Then Chapters 8 and 9 can be read in either order.
- Analysis: Chapter 5 introduces some tools that are necessary for analyzing algorithms. It uses proof by induction, which is discussed in Section 4.4.
- Algebra: Chapter 10 uses recursively defined functions as discussed in Section 3.2, and it uses proof by induction as discussed in Section 4.4.

Course Suggestions

The topics in the book can be presented in a variety of ways, depending on the length of the course, the emphasis, and student background. The major portion of the text has been taught for several years to sophomores at Portland State University. Here are a few suggestions for courses of various lengths and emphases. (Assume that a lecture is a 50-minute period.)

Discrete mathematics with minimal formal logic

Chapters or Sections 1–4, 5.1–5.4, 6.2, 7.1, 10.1–10.2. Pace: About 40 to 45 lectures for an average-pace course. A course of 60 lectures might go at a slower pace or add some additional sections.

Discrete mathematics with some formal logic

Chapters or Sections 1–5, 6.2–6.3, 7, 8.1–8.2, 10.1–10.2, and 10.5.1–10.5.2. Pace: About 60 lectures for an average-pace course. A course of 80 lectures might go at a slower pace or add some additional sections.

Discrete mathematics and formal logic (a full course)

Chapters: 1–10. Pace: About 90 lectures for an average-pace course.

Logic

Chapters 6-9 with references to Sections 1.1, 1.2, 2.1, and 4.4. Pace: About 40 to 45 lectures for an average-pace course. A slower-pace course of 60 lectures might spend more time on 1.1 and 4.4.

Acknowledgments

Many people helped me create this book in the first place and many people have influenced the content and form of the second edition. Thanks go especially to the students and teachers who have kept my e-mail quite interesting over the past several years with questions, suggestions, and criticisms. They have all influenced this book. At Jones and Bartlett, I thank Michael Stranz for maintaining the entrepreneurial spirit, and I thank Amy Rose for overseeing the production. I also thank my family for their constant help and support.

J.L.H.
Portland, Oregon

Contents

chapter 1

Elementary Notions and Notations

'Excellent!' I cried. 'Elementary,' said he.
—Watson in *The Crooked Man*
by Arthur Conan Doyle (1859–1930)

To communicate, we sometimes need to agree on the meaning of certain terms. If the same idea is mentioned several times in a discussion, we often replace it with some shorthand notation. The choice of notation can help us avoid wordiness and ambiguity, and it can help us achieve conciseness and clarity in our written and oral expression.

Many problems of computer science, as well as other areas of thought, deal with reasoning about things and representing things. Since much of our communication involves reasoning about things, we'll begin the chapter with a short discussion about the notions of informal proof. The rest of the chapter is devoted to introducing the basic notions and notations for sets, tuples, graphs, and trees. The treatment here is introductory in nature, and we'll expand on these ideas in later chapters as the need arises.

chapter guide

Section 1.1 introduces some proof techniques that are used throughout the book. We'll practice each technique with a proof about numbers.

Section 1.2 introduces the basic ideas about sets. We'll see how to compare them and how to combine them, and we'll introduce some elementary ways to count them. We'll also introduce bags, which are like sets but which might contain repeated occurrences of elements, and we'll have a little discussion about why we should stick with uncomplicated sets.

Section 1.3 introduces some basic ideas about ordered structures and how to represent them. Tuples are introduced as a notation for ordered information. We'll introduce the notions and notations for lists, strings, and relations. We'll also see some elementary ways to count tuples.

Section 1.4 introduces the basic ideas about graphs and trees. We'll discuss ways to represent them, ways to traverse them, and we'll see a famous algorithm for constructing a spanning tree for a graph.

1.1 A Proof Primer

For our purposes an *informal proof* is a demonstration that some statement is true. We normally communicate an informal proof in an English-like language that mixes everyday English with symbols that appear in the statement to be proved. In the next few paragraphs we'll discuss some basic techniques for doing informal proofs. These techniques will come in handy in trying to understand someone's proof or in trying to construct a proof of your own, so keep them in your mental tool kit.

We'll start off with a short refresher on logical statements followed by a short discussion about numbers. This will give us something to talk about when we look at examples of informal proof techniques.

1.1.1 Logical Statements

For this primer we'll consider only statements that are either true or false. We'll start by discussing some familiar ways to structure logical statements.

Negation

If S represents some statement, then the *negation* of S is the statement "not S," whose truth value is opposite that of S. We can represent this relationship with a *truth table* in which each row gives a value for S and the corresponding value for not S:

S	not S
true	false
false	true

We often paraphrase the negation of a statement to make it more understandable. For example, to negate the statement "Earth is a star," we normally say, "Earth is not a star," or "It is not the case that Earth is a star," rather than "Not Earth is a star."

We should also observe that negation relates statements about *every* case with statements about *some* case. For example, the statement "Not every planet has a moon" has the same meaning as "Some planet does not have a moon." Similarly, the statement "It is not the case that some planet is a star" has the same meaning as "Every planet is not a star."

A	*B*	*A* and *B*	*A* or *B*
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Figure 1.1 Truth tables.

Conjunction and Disjunction

The *conjunction* of A and B is the statement "A and B," which is true when both A and B are true. The *disjunction* of A and B is the statement "A or B," which is true if either or both of A and B are true. The truth tables for conjunction and disjunction are given in Figure 1.1.

Sometimes we paraphrase conjunctions and disjunctions. For example, instead of "Earth is a planet and Mars is a planet," we might write "Earth and Mars are planets." Instead of "x is positive or y is positive," we might write "Either x or y is positive."

Conditional Statements

Many statements are written in the general form "If A then B," where A and B are also logical statements. Such a statement is called a *conditional statement* in which A is the *hypothesis* and B is the *conclusion*. We can read "If A then B" in several other ways: "A is a sufficient condition for B," or "B is a necessary condition for A," or simply "A implies B." The truth table for a conditional statement is contained in Figure 1.2.

Let's make a few comments about this table. Notice that the conditional is false only when the hypothesis is true and the conclusion is false. It's true in the other three cases. The conditional truth table gives some people fits because they interpret "If A then B" to mean "B can be proved from A," which assumes that A and B are related in some way. But we've all heard statements like "If the moon is made of green cheese, then $1 = 2$." We nod our heads and agree that the statement is true, even though there is no relationship between the hypothesis and conclusion. Similarly, we shake our heads and don't agree with a statement like "If $1 = 1$, then the moon is made of green cheese."

A	*B*	if *A* then *B*
true	true	true
true	false	false
false	true	true
false	false	true

Figure 1.2 Truth table.

When the hypothesis of a conditional is false, we say that the conditional is *vacuously* true. For example, the statement "If $1 = 2$, then $39 = 12$" is vacuously true because the hypothesis is false. If the conclusion is true, we say that the conditional is *trivially* true. For example, the statement "If $1 = 2$, then $2 + 2 = 4$" is trivially true because the conclusion is true. We leave it to the reader to convince at least one person that the conditional truth table is defined properly.

The *converse* of "If A then B" is "If B then A." The converse does not always have the same truth value. For example, we know that the following statement about numbers is true.

$$\text{If } x > 0 \text{ and } y > 0, \text{ then } x + y > 0.$$

The converse of this statement is

$$\text{If } x + y > 0, \text{ then } x > 0 \text{ and } y > 0.$$

This converse is false. For example, let $x = 3$ and $y = -2$. Then the statement becomes "If $3 + (-2) > 0$, then $3 > 0$ and $-2 > 0$," which is false.

Equivalent Statements

Sometimes it's convenient to write a statement in a different form but with the same truth value. Two statements are said to be *equivalent* if they have the same truth value for any assignment of truth values to the variables that occur in the statements.

We can combine negation with either conjunction or disjunction to obtain the following pairs of equivalent statements.

"not (A and B)" is equivalent to "(not A) or (not B)."
"not (A or B)" is equivalent to "(not A) and (not B)."

For example, the statement "not ($x > 0$ and $y > 0$)" is equivalent to the statement "$x \leq 0$ or $y \leq 0$." The statement "not ($x > 0$ or $y > 0$)" is equivalent to the statement "$x \leq 0$ and $y \leq 0$."

Conjunctions and disjunctions distribute over each other in the following sense:

"A and (B or C)" is equivalent to "(A and B) or (A and C)."
"A or (B and C)" is equivalent to "(A or B) and (A or C)."

For example, the statement "$0 < x$ and ($x < 4$ or $x < 9$)" is equivalent to the statement "$0 < x < 4$ or $0 < x < 9$." The statement "$x > 0$ or ($x > 2$ and $x > 1$)" is equivalent to "($x > 0$ or $x > 2$) and ($x > 0$ or $x > 1$)."

The *contrapositive* of the conditional statement "If A then B" is the equivalent statement "If not B then not A." For example, the statement

$$\text{If (x > 0 and y > 0), then x + y > 0}$$

is equivalent to the statement

If $x + y \leq 0$, then ($x \leq 0$ or $y \leq 0$).

We can also express the conditional statement "If A, then B" in terms of the equivalent statement "(not A) or B." For example, the statement

If $x > 0$ and $y > 0$, then $x + y > 0$.

is equivalent to the statement

($x \leq 0$ or $y \leq 0$) or $x + y > 0$.

Since we can express a conditional in terms of negation and disjunction, it follows that the statements "not (If A then B)" and "A and (not B)" are equivalent. For example, the statement

It is not the case that if Earth is a planet, then Earth is a star.

is equivalent to the statement

Earth is a planet and Earth is not a star.

Let's summarize the equivalences that we have discussed. Each row of the following table contains two equivalent statements S and T.

S (is equivalent to)	T
not (A and B)	(not A) or (not B)
not (A or B)	(not A) and (not B)
A and (B or C)	(A and B) or (A and C)
A or (B and C)	(A or B) and (A or C)
if A then B	if not B then not A
if A then B	(not A) or B
not (if A then B)	A and (not B)

1.1.2 Something to Talk About

To discuss proof techniques, we need something to talk about when giving sample proofs. Since numbers are familiar to everyone, that's what we'll talk about. But to make sure that we all start on the same page, we'll review a little terminology.

The numbers that we'll be discussing are called *integers*, and we can list them as follows:

$$\ldots, -4, -3, -2, -1, 0, 1, 2, 3, 4, \ldots .$$

The integers in the following list are called *even* integers:

$$\ldots, -4, -2, 0, 2, 4, \ldots .$$

The integers in the following list are called *odd* integers:

$$\ldots, -3, -1, 1, 3, \ldots .$$

So every integer is either even or odd but not both. In fact, every even integer has the form $2n$ for some integer n. Similarly, every odd integer has the form $2n + 1$ for some integer n.

Divisibility and Prime Numbers

An integer d *divides* an integer n if $d \neq 0$ and there is an integer k such that $n = dk$. For example, 3 divides 18 because we can write $18 = (3)(6)$. But 5 does not divide 18 because there is no integer k such that $18 = 5k$. The following list shows all the divisors of 18.

$$-18, -9, -6, -3, -2, -1, 1, 2, 3, 6, 9, 18.$$

Some alternative words for *d divides n* are *d is a divisor of n* or *n is divisible by d.* We often denote the fact that d divides n with the following shorthand notation:

$$d|n$$

For example, we have $-9|9$, $-3|9$, $-1|9$, $1|9$, $3|9$, and $9|9$. Here are two properties of divisibility that we'll record for future use.

Divisibility Properties **(1.1)**

a. If $d|a$ and $a|b$, then $d|b$.

b. If $d|a$ and $d|b$, then $d|(ax + by)$ for any integers x and y.

An integer $p > 1$ is called a *prime* number if 1 and p are its only positive divisors. For example, the first eight prime numbers are

$$2, 3, 5, 7, 11, 13, 17, 19.$$

Prime numbers have many important properties and they have many applications in computer science. But for now all we need to know is the definiton of a prime.

1.1.3 Proof Techniques

Now that we have something to talk about, we'll discuss some fundamental proof techniques and give some sample proofs for each technique.

Proof by Exhaustive Checking

When a statement asserts that each of a finite number of things has a certain property, then we might be able to prove the statement by checking that each thing has the stated property. For example, suppose someone says, "If n is an integer and $2 \leq n \leq 7$, then $n^2 + 2$ is not divisible by 4." We can prove the statement by *exhaustive checking.* For $2 \leq n \leq 7$, the corresponding values of $n^2 + 2$ are

$$6, 11, 18, 27, 38, 51.$$

We can check that these numbers are not divisible by 4. For another example, suppose someone says, "If n is an integer and $2 \leq n \leq 500$, then $n^2 + 2$ is not divisible by 4." Again, this statement can be proved by exhaustive checking, but perhaps by a computer rather than a person.

Exhaustive checking cannot be used to prove a statement that requires infinitely many things to check. For example, consider the statement, "If n is an integer, then $n^2 + 2$ is not divisible by 4." This statement is true, but there are infinitely many things to check. So another proof technique will be required. We'll get to it after a few more paragraphs.

An example that proves a statement false is often called a *counterexample.* Sometimes counterexamples can be found by exhaustive checking. For example, consider the statement, "Every odd number greater than 1 that is not prime has the form $2 + p$ for some prime p." We can observe that the statement is false because 27 is a counterexample.

Conditional Proof

Many statements that we wish to prove are in conditional form or can be phrased in conditional form (if A then B). The *direct approach* to proving such a statement starts with the assumption that the hypothesis A is true. The next step is to find a statement that is implied by the assumption or known facts. Each step proceeds in this fashion to find a statement that is implied by any of the previous statements or known facts. The *conditional proof* ends when the conclusion B is reached.

example 1.1 A Proof About Sums

We'll prove the following general statement about integers:

The sum of any two odd integers is an even integer.

We can rephrase the statement in the conditional form

If x and y are odd integers, then $x + y$ is an even integer.

Proof: Assume the hypothesis that x and y are odd integers. It follows that x and y can be written in the the form $x = 2k + 1$ and $y = 2m + 1$, where k and m are arbitrary integers. Now substitute for x and y in $x + y$ to obtain

$$x + y = (2k + 1) + (2m + 1) = 2k + 2m + 2 = 2(k + m + 1).$$

Since the expression on the right-hand side contains 2 as a factor, it represents an even integer. QED.

end example

example **1.2 A Divisibility Proof**

We'll prove the following statement (1.1a) about divisibility:

If $d|a$ and $a|b$, then $d|b$.

Proof: Assume the hypothesis that $d|a$ and $a|b$. It follows from the definition of divisibility that there are integers m and n such that $a = dm$ and $b = an$. Now substitute for a in the second equation.

$$b = an = (dm)n = d(mn).$$

This equation says that $d|b$. QED.

end example

Proving the Contrapositive

Recall that a conditional statement "if A then B" and its contrapositive "if not B then not A" have the same truth table. So a proof of one is also a proof of the other. The *indirect approach* to proving "if A then B" is to *prove the contrapositve.* Start with the assumption that B is false. The next step is to find a statement that is implied by the assumption or known facts. Each step proceeds in this fashion to find a statement that is implied by any of the previous statements or known facts. The proof ends when an implied statement says that A is false.

example **1.3 An Odd Proof**

We'll prove the following statement about the integers:

If x^2 is odd, then x is odd.

To prove the statement, we'll prove its contrapositive:

If x is even, then x^2 is even.

Proof: Assume the hypothesis (of the contrapositive) that x is even. It follows that $x = 2k$ for some integer k. Now square x and substitute for x to obtain

$$x^2 = (2k)^2 = 4k^2 = 2(2k^2).$$

The expression on the right side of the equation represents an even number. Therefore x^2 is even. QED.

end example

Proof by Contradiction

A *contradiction* is a false statement. Another kind of indirect proof is *proof by contradiction*, where we start out by assuming that the statement to be proved is false. Then we argue until we reach a contradiction. Such an argument is often called a *refutation.*

Proof by contradiction is often the method of choice because we can wander wherever the proof takes us to find a contradiction. We'll give two examples to show the wandering that can take place.

example 1.4 A Not-Divisible Proof

We'll prove the following statement about divisibility:

If n is an integer, then $n^2 + 2$ is not divisible by 4.

Proof: Assume the statement is false. Then $4 \mid (n^2 + 2)$ for some integer n. This means that $n^2 + 2 = 4k$ for some integer k. We'll consider the two cases where n is even and where n is odd. If n is even, then $n = 2m$ for some integer m. Substituting for n we obtain

$$4k = n^2 + 2 = (2m)^2 + 2 = 4m^2 + 2.$$

We can divide both sides of the equation by 2 to obtain

$$2k = 2m^2 + 1.$$

This says that an even number $(2k)$ is equal to an odd number $(2m^2 + 1)$, which is a contradiction. Therefore, n cannot be even. Now assume n is odd. Then $n = 2m + 1$ for some integer m. Substituting for n we obtain

$$4k = n^2 + 2 = (2m + 1)^2 + 2 = 4m^2 + 4m + 3.$$

Isolate 3 on the right side of the equation to obtain

$$4k - 4m^2 - 4m = 3.$$

This is a contradiction because the left side is even and the right side is odd. Therefore, n cannot be odd. QED.

end example

example

1.5 Prime Numbers

We'll prove the following statement about integers:

Every integer greater than 1 is divisible by a prime.

Proof: Assume the statement is false. Then some integer $n > 1$ is not divisible by a prime. Since a prime divides itself, n cannot be a prime. So there is at least one integer d such that $d|n$ and $1 < d < n$. Assume that d is the smallest divisior of n between 1 and n. Now d is not prime, or else it would be a prime divisor of n. So there is an integer a such that $a|d$ and $1 < a < d$. Since $a \mid d$ and $d \mid n$, it follows from (1.1a) that $a \mid n$. But now we have $a \mid n$ and $1 < a < d$, which contradicts the assumption that d is the smallest such divisor of n. QED.

end example

If and Only If Proofs

The statement "A if and only if B" is shorthand for the two statements "If A then B" and "If B then A." The abbreviation "A *iff* B" is often used for "A if and only if B." Instead of "A iff B," some people write "A is a necessary and sufficient condition for B" or "B is a necessary and sufficient condition for A." Remember that two proofs are required for an iff statement, one for each conditional statement.

example

1.6 An Iff Proof

We'll prove the following iff statement about integers:

$$x \text{ is odd if and only if } 8 \mid (x^2 - 1).$$

To prove this iff statement, we must prove the following two statements:

a. If x is odd, then $8 \mid (x^2 - 1)$.

b. If $8 \mid (x^2 - 1)$, then x is odd.

Proof of (a): Assume x is odd. Then we can write x in the form $x = 2k + 1$ for some integer k. Substituting for x in $x^2 - 1$ gives

$$x^2 - 1 = 4k^2 + 4k = 4k(k + 1).$$

Since k and $k + 1$ are consecutive integers, one is odd and the other is even, so the product $k(k + 1)$ is even. So $k(k + 1) = 2m$ for some integer m. Substituting for $k(k + 1)$ gives

$$x^2 - 1 = 4k(k + 1) = 4(2m) = 8m.$$

Therefore, $8 \mid (x^2 - 1)$, so part (a) is proven.

Proof of (b): Assume $8 \mid (x^2 - 1)$. Then $x^2 - 1 = 8k$ for some integer k. Therefore, we have $x^2 = 8k + 1 = 2(4\text{k}) + 1$, which has the form of an odd integer. So x^2 is odd, and it follows from Example 1.3 that x is odd, so part (b) is proven. Therefore, the iff statement is proven. QED.

end example

Sometimes we encounter iff statements that can be proven by using statements that are related to each other by iff. Then a proof can be constructed as a sequence of iff statements. For example, to prove A iff B we might be able to find a statement C such that A iff C and C iff B are both true. Then we can conclude that A iff B is true. The proof could then be put in the form A iff C iff B.

example **1.7 Two Proofs in One**

We'll prove the following statement about integers:

$$x \text{ is odd if and only if } x^2 + 2x + 1 \text{ is even.}$$

Proof: The following sequence of iff statements connects the left side to the right side. (The reason for each step is given in parentheses.)

$$\begin{array}{lll}
x \text{ is odd iff} & x = 2k + 1 \text{ for some integer } k & \text{(definition)} \\
\text{iff} & x + 1 = 2k + 2 \text{ for some integer } k & \text{(algebra)} \\
\text{iff} & x + 1 = 2m \text{ for some integer } m & \text{(algebra)} \\
\text{iff} & x + 1 \text{ is even} & \text{(definition)} \\
\text{iff} & (x + 1)^2 \text{ is even} & \text{(Exercise 8a)} \\
\text{iff} & x^2 + 2x + 1 \text{ is even} & \text{(algebra)} \quad \text{QED}
\end{array}$$

end example

On Constructive Existence

If a statement asserts that some object exists, then we can try to prove the statement in either of two ways. One way is to use proof by contradiction, in which we assume that the object does not exist and then come up with some kind of contradiction. The second way is to construct an instance of the object. In either case we know that the object exists, but the second way also gives us an instance of the object. Computer science leans toward the construction of objects by algorithms. So the *constructive approach* is usually preferred, although it's not always possible.

Important Note

Always try to write out your proofs. Use complete sentences that describe your reasoning. If your proof seems to consist only of a bunch of equations or expressions, you still need to describe how they contribute to the proof. Try to write your proofs the same way you would write a letter to a friend who wants to understand what you have written.

Exercises

1. See whether you can convince yourself, or a friend, that the conditional truth table is correct by making up English sentences of the form "If A then B."

2. Verify that the truth tables for each of the following pairs of statements are identical.
   a. "not (A and B)" and "(not A) or (not B)."
   b. "not (A or B)" and "(not A) and (not B)."
   c. "if A, then B" and "if (not B), then (not A)."
   d. "if A, then B" and "(not A) or B."
   e. "not (if A then B)" and "A and (not B)."

3. Prove or disprove each of the following statements by exhaustive checking.
   a. There is a prime number between 45 and 54.
   b. The product of any two of the four numbers 2, 3, 4, and 5 is even.
   c. Every odd integer between 2 and 26 is either prime or the product of two primes.
   d. If $d \mid ab$, then $d \mid a$ or $d \mid b$.
   e. If m and n are integers, then $(3m + 2)(3n + 2)$ has the form $(3k + 2)$ for some integer k.

4. Prove each of the following statements about the integers.
   a. If x and y are even, then $x + y$ is even.
   b. If x is even and y is odd, then $x + y$ is odd.
   c. If x and y are odd, then $x - y$ is even.
   d. If $3n$ is even, then n is even.

5. Write down the converse of the following statement about integers:

   If x and y are odd, then $x - y$ is even.

   Is the statement that you wrote down true or false? Prove your answer.

6. Prove each of the following statements, where m and n are integers.
   a. If $x = 3m + 4$ and $y = 3n + 4$, then $xy = 3k + 4$ for some integer k.
   b. If $x = 5m + 6$ and $y = 5n + 6$, then $xy = 5k + 6$ for some integer k.
   c. If $x = 7m + 8$ and $y = 7n + 8$, then $xy = 7k + 8$ for some integer k.
7. Prove each of the following statements about divisibility of integers.
   a. If $d|(da + b)$, then $d|b$.
   b. If $d \mid (a + b)$ and $d|a$, then $d|b$.
   c. (1.1b) If $d|a$ and $d|b$, then $d|(ax + by)$ for any integers x and y.
8. Prove each of the following iff statements about integers.
   a. x is even if and only if x^2 is even.
   b. xy is odd if and only if x is odd and y is odd.
   c. x is odd if and only if $x^2 + 6x + 9$ is even.
   d. $m \mid n$ and $n \mid m$ if and only if $n = m$ or $n = -m$.

1.2 Sets

In our everyday discourse we sometimes run into the problem of trying to define a word in terms of other words whose definitions may include the word we are trying to define. That's the problem we have in trying to define the word *set*. To illustrate the point, we often think of some (perhaps all) of the words

set, collection, bunch, group, class

as synonyms for each other. We pick up the meaning for such a word intuitively by seeing how it is used.

1.2.1 Definition of a Set

We'll simply say that a *set* is a collection of things called its *elements*, *members*, or *objects*. Sometimes the word *collection* is used in place of *set* to clarify a sentence. For example, "a collection of sets" seems clearer than "a set of sets." We say that a set contains its elements, or that the elements belong to the set, or that the elements are in the set. If S is a set and x is an element in S, then we write

$$x \in S.$$

If x is not an element of S, then we write $x \notin S$. If $x \in S$ and $y \in S$, we often denote this fact by the shorthand notation

$$x, y \in S.$$

Describing Sets

To describe a set we need to describe its elements in some way. One way to define a set is to explicitly name its elements. A set defined in this way is denoted by listing its elements, separated by commas, and surrounding the listing with braces. For example, the set S consisting of the letters x, y, and z is denoted by

$$S = \{x, y, z\}.$$

Sets can have other sets as elements. For example, the set $A = \{x, \{x, y\}\}$ has two elements. One element is x, and the other element is $\{x, y\}$. So we can write $x \in A$ and $\{x, y\} \in$ A.

An important characteristic of sets is that there are no repeated occurrences of elements. For example, $\{H, E, L, L, O\}$ is not a set since there are two occurrences of the letter L.

We often use the three-dot ellipsis, $\ldots$, to informally denote a sequence of elements that we do not wish to write down. For example, the set

$$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$$

can be denoted in several different ways with ellipses, two of which are

$$\{1, 2, \ldots, 12\} \text{ and } \{1, 2, 3, \ldots, 11, 12\}.$$

The set with no elements is called the *empty set*—some people refer to it as the *null set.* The empty set is denoted by { } or more often by the symbol

$$\varnothing.$$

A set with one element is called a *singleton.* For example, $\{a\}$ and $\{b\}$ are singletons.

Equality of Sets

Two sets are *equal* if they have the same elements. We denote the fact that two sets A and B are equal by writing

$$A = B.$$

An important characteristic of sets is that there is no particular order or arrangement of the elements. For example, the set whose elements are g, h, and u can be represented in many different ways, two of which are

$$\{u, g, h\} = \{h, u, g\}.$$

If the sets A and B are not equal, we write

$$A \neq B.$$

For example, $\{a, b, c\} \neq \{a, b\}$ because c is an element of only one of the sets. We also have $\{a\} \neq \varnothing$ because the empty set doesn't have any elements.

Before we go any further let's record the two important characteristics of sets that we have discussed.

Two Characteristics of Sets

1. There are no repeated occurrences of elements.
2. There is no particular order or arrangement of the elements.

Finite and Infinite Sets

Suppose we start counting the elements of a set S one element per second of time with a stop watch. If $S = \varnothing$, then we don't need to start, because there are no elements to count. But if $S \neq \varnothing$, we agree to start the counting after we have started the timer. If a point in time is reached when all the elements of S have been counted, then we stop the timer, or in some cases we might need to have one of our descendants stop the timer. In this case we say that S is a *finite* set. If the counting never stops, then S is an *infinite* set. All the examples that we have discussed to this point are finite sets. We will discuss counting finite and infinite sets in other parts of the book as the need arises.

Natural Numbers and Integers

Familiar infinite sets are sometimes denoted by listing a few of the elements followed by an ellipsis. We reserve some letters to denote specific sets that we'll refer to throughout the book. For example, the set of *natural numbers* will be denoted by $\mathbb{N}$[1] and the set of *integers* by $\mathbb{Z}$. So we can write

$$\mathbb{N} = \{0, 1, 2, 3, \ldots\} \quad \text{and} \quad \mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}.$$

Describing Sets by Properties

Many sets are hard to describe by listing elements. Examples that come to mind are the *rational numbers*, which we denote by $\mathbb{Q}$, and the *real numbers*, which we denote by $\mathbb{R}$. Instead of listing the elements, we can often describe a property that the elements of the set satisfy. For example, the set of odd integers consists of integers having the form $2k + 1$ for some integer k.

[1]Some people consider the natural numbers to be the set $\{1, 2, 3, \ldots\}$. If you are one of these people, then think of $\mathbb{N}$ as the nonnegative integers.

If P is a property, then the set S whose elements have property P is denoted by writing

$$S = \{x \mid x \text{ has property } P\}.$$

We read this as "S is the set of all x such that x has property P." For example, if we let *Odd* be the set of odd integers, then we can describe *Odd* in several ways.

$$\begin{aligned} Odd &= \{\ldots, -5, -3, -1, 1, 3, 5, \ldots\} \\ &= \{x \mid x \text{ is an odd integer}\} \\ &= \{x \mid x = 2k + 1 \text{ for some integer } k\} \\ &= \{x \mid x = 2k + 1 \text{ for some } k \in \mathbb{Z}\}. \end{aligned}$$

Of course, we can also describe finite sets by finding properties that they possess. For example,

$$\{1, 2, \ldots, 12\} = \{x \mid x \in \mathbb{N} \text{ and } 1 \leq x \leq 12\}.$$

We can also describe a set by writing expressions for the elements. For example, the set *Odd* has the following additional descriptions.

$$\begin{aligned} Odd &= \{2k + 1 \mid k \text{ is an integer}\}. \\ &= \{2k + 1 \mid k \in \mathbb{Z}\}. \end{aligned}$$

Subsets

If A and B are sets and every element of A is also an element of B, then we say that A is a *subset* of B and write

$$A \subset B.$$

For example, we have $\{a, b\} \subset \{a, b, c\}$, $\{0, 1, 2\} \subset \mathbb{N}$, and $\mathbb{N} \subset \mathbb{Z}$. It follows from the definition that every set A is a subset of itself. Thus we have $A \subset A$. It also follows from the definition that the empty set is a subset of any set A. So we have $\varnothing \subset A$. Can you see why? We'll leave this as an exercise.

If $A \subset B$ and there is some element in B that does not occur in A, then A is called a *proper* subset of B. For example, $\{a, b\}$ is a proper subset of $\{a, b, c\}$. We also conclude that $\mathbb{N}$ is a proper subset of $\mathbb{Z}$, $\mathbb{Z}$ is a proper subset of $\mathbb{Q}$, and $\mathbb{Q}$ is a proper subset of $\mathbb{R}$.

If A is not a subset of B, we sometimes write

$$A \not\subset B.$$

For example, $\{a, b\} \not\subset \{a, c\}$ and $\{0, -1, -2\} \not\subset \mathbb{N}$. Remember that the idea of subset is different from the idea of membership. For example, if $A = \{a, b, c\}$, then $\{a\} \subset A$ and $a \in A$. But $\{a\} \notin A$ and $a \not\subset A$. For another example, let $A = \{a, \{b\}\}$. Then $a \in A$, $\{b\} \in A$, $\{a\} \subset A$, and $\{\{b\}\} \subset A$. But $b \notin A$ and $\{b\} \not\subset A$.

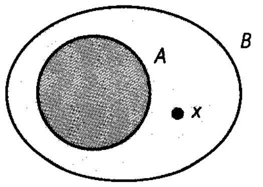


Figure 1.3 Venn diagram of proper subset $A \subset B$.

The Power Set

The collection of all subsets of a set S is called the *power set* of S, which we denote by power(S). For example, if $S = \{a, b, c\}$, then the power set of S can be written as follows:

$$\text{power}(S) = \{\varnothing\ , \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, S\}.$$

An interesting programming problem is to construct the power set of a finite set. We'll discuss this problem later, once we've developed some tools to help build an easy solution.

Venn Diagrams

In dealing with sets, it's often useful to draw a picture in order to visualize the situation. A *Venn diagram*—named after the logician John Venn (1834–1923)—consists of one or more closed curves in which the interior of each curve represents a set. For example, the Venn diagram in Figure 1.3 represents the fact that A is a proper subset of B and x is an element of B that does not occur in A.

Proof Strategies with Subsets and Equality

Subsets allow us to give a precise definition of set equality: Two sets are equal if they are subsets of each other. In more concise form we can write

Equality of Sets (1.2)

$$A = B \text{ means } A \subset B \text{ and } B \subset A$$

Let's record three useful proof strategies for comparing two sets.

Statement to Prove	*Proof Strategy*
$A \subset B$	For arbitrary $x \in A$, show that $x \in B$.
$A \not\subset B$	Find an element $x \in A$ such that $x \notin B$.
$A = B$	Show that $A \subset B$ and show that $B \subset A$.

example **1.8 Subset Proof**

We'll show that $A \subset B$, where A and B are defined as follows:

$$A = \{x \mid x \text{ is a prime number and } 42 \leq x \leq 51\},$$
$$B = \{x \mid x = 4k + 3 \text{ and } k \in \mathbb{N}\}.$$

We start the proof by letting $x \in A$. Then either $x = 43$ or $x = 47$. We can write $43 = 4(10) + 3$ and $47 = 4(11) + 3$. So in either case, x has the form of an element of B. Thus $x \in B$. Therefore, $A \subset B$.

end example

example **1.9 Not-Subset Proof**

We'll show that $A \not\subset B$ and $B \not\subset A$, where A and B are defined by

$$A = \{3k + 1 \mid k \in \mathbb{N}\} \text{ and } B = \{4k + 1 \mid k \in \mathbb{N}\}.$$

By listing a few elements from each set we can write A and B as follows:

$$A = \{1, 4, 7, ...\} \text{ and } B = \{1, 5, 9, ...\}.$$

Now it's easy to prove that $A \not\subset B$ because $4 \in A$ and $4 \notin B$. We can also prove that $B \not\subset A$ by observing that $5 \in B$ and $5 \notin A$.

end example

example **1.10 Equal Sets Proof**

We'll show that $A = B$, where A and B are defined as follows:

$$A = \{x \mid x \text{ is prime and } 12 \leq x \leq 18\},$$
$$B = \{x \mid x = 4k + 1 \text{ and } k \in \{3, 4\}\}.$$

First we'll show that $A \subset B$. Let $x \in A$. Then either $x = 13$ or $x = 17$. We can write $13 = 4(3) + 1$ and $17 = 4(4) + 1$. It follows that $x \in B$. Therefore, $A \subset B$. Next we'll show that $B \subset A$. Let $x \in B$. It follows that either $x = 4(3) + 1$ or $x = 4(4) + 1$. In either case, x is a prime number between 12 and 18. Therefore, $B \subset A$. So $A = B$.

end example

1.2.2 Operations on Sets

We'll discuss the operations of union, intersection, and complement, all of which combine sets to form new sets.

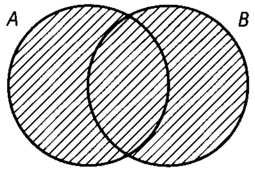


Figure 1.4 Venn diagram of $A \cup B$.

Union of Sets

The *union* of two sets A and B is the set of all elements that are either in A or in B or in both A and B. The union is denoted by $A \cup B$ and we can give the following formal definition.

Union of Sets **(1.3)**

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

The use of the word "or" in the definition is taken to mean "either or both." For example, if $A = \{a, b, c\}$ and $B = \{c, d\}$, then $A \cup B = \{a, b, c, d\}$. The union of two sets A and B is represented by the shaded regions of the Venn diagram in Figure 1.4.

The following properties give some basic facts about the union operation.

Properties of Union **(1.4)**

a. $A \cup \varnothing = A$.
b. $A \cup B = B \cup A$. ($\cup$ is commutative.)
c. $A \cup (B \cup C) = (A \cup B) \cup C$. ($\cup$ is associative.)
d. $A \cup A = A$.
e. $A \subset B$ if and only if $A \cup B = B$.

example 1.11 A Subset Condition

We'll prove the following statement (1.4e) and leave the other parts as exercises.

$$A \subset B \text{ if and only if } A \cup B = B.$$

Proof: Since this is an if and only if statement we have two statements to prove. First we'll prove that $A \subset B$ implies $A \cup B = B$. Assume that $A \subset B$. With this assumption we must show that $A \cup B = B$. Let $x \in A \cup B$. It follows that $x \in A$ or $x \in B$. Since we have assumed that $A \subset B$, it follows that $x \in B$. Thus $A \cup B \subset B$. But since we always have $B \subset A \cup B$, it follows from (1.2) that

$A \cup B = B$. So the first part is proven. Next we'll prove that $A \cup B = B$ implies $A \subset B$. Assume that $A \cup B = B$. If $x \in A$, then $x \in A \cup B$. Since we are assuming that $A \cup B = B$, it follows that $x \in B$. Therefore $A \subset B$. So the second part is proven. QED.

end example

The union operation can be defined for an arbitrary collection of sets in a natural way. For example, the union of the n sets $A_1, \ldots, A_n$ can be denoted in the following way:

$$\bigcup_{i=1}^{n} A_i = A_1 \cup \cdots \cup A_n.$$

The union of the infinite collection of sets $A_1, A_2, \ldots, A_n, \ldots$ can be denoted in the following way.

$$\bigcup_{i=1}^{\infty} A_i = A_1 \cup \cdots \cup A_n \cup \cdots .$$

If I is a set of indices and A_i is a set for each $i \in I$, then the union of the sets in the collection can be denoted in the following way:

$$\bigcup_{i \in I} A_i.$$

example 1.12 English Words as a Union

Let W be the set of all words in the English language. Then we can represent W as an infinite union of sets. For each $i > 0$, let A_i denote the set of all words with i letters. Then W has the following representation:

$$W = \bigcup_{i=1}^{\infty} A_i.$$

If no English word has more than 25 letters, then $A_i = \varnothing$ for $i > 25$. In this case we could write W as the finite union

$$W = \bigcup_{i=1}^{25} A_i.$$

end example

example **1.13 An Infinite Union of Finite Sets**

Suppose we want to calculate the union of the sets $A_i = \{-2i, 2i\}$ where i is an odd natural number. For example, $A_3 = \{-6, 6\}$ and $A_5 = \{10, -10\}$. If we let *Odd* be the set of odd natural numbers, then we can write

$$\bigcup_{i \in Odd} A_i = \{\ldots, -10, -6, -2, 2, 6, 10, \ldots\}.$$

end example

Intersection of Sets

The *intersection* of two sets A and B is the set of all elements that are in both A and B. The intersection is denoted by $A \cap B$ and we can give the following formal definition.

Intersection of Sets **(1.5)**

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

For example, if $A = \{a, b, c\}$ and $B = \{c, d\}$, then $A \cap B = \{c\}$. If $A \cap B = \varnothing$, then A and B are said to be *disjoint.* The nonempty intersection of two sets A and B is represented by the shaded region of the Venn diagram in Figure 1.5.

The following properties give some basic facts about the intersection operation.

Properties of Intersection **(1.6)**

a. $A \cap \varnothing = \varnothing$.

b. $A \cap B = B \cap A$. ($\cap$ is commutative.)

c. $A \cap (B \cap C) = (A \cap B) \cap C$. ($\cap$ is associative.)

d. $A \cap A = A$.

e. $A \subset B$ if and only if $A \cap B = A$.

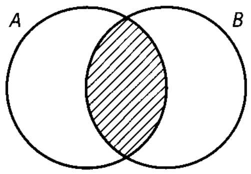


Figure 1.5 Venn diagram of $A \cap B$.

The intersection operation can be defined for an arbitrary collection of sets in a natural way. For example, the intersection of the n sets $A_1, \ldots, A_n$ can be denoted in the following way:

$$\bigcap_{i=1}^{n} A_i = A_1 \cap \cdots \cap A_n.$$

The intersection of the infinite collection of sets $A_1, A_2, \ldots, A_n, \ldots$ can be denoted in the following way:

$$\bigcap_{i=1}^{\infty} A_i = A_1 \cap \cdots \cap A_n \cap \cdots .$$

If I is a set of indices and A_i is a set for each $i \in I$, then the intersection of the sets in the collection can be denoted in the following way:

$$\bigcap_{i \in I} A_i.$$

example **1.14 An Infinite Intersection**

For each odd natural number i let $A_i = \{x \mid x \in \mathbb{N} \text{ and } -i \leq x \leq i\}$. For example, $A_3 = \{-3, -2, -1, 0, 1, 2, 3\}$. Letting *Odd* be the set of odd natural numbers, we can represent the intersection of the sets A_i as

$$\bigcap_{i \in Odd} A_i = \{-1, 0, 1\}.$$

end example

Difference of Sets

If A and B are sets, then the *difference* $A - B$ (also called the *relative complement* of B in A) is the set of elements in A that are not in B, which we can describe as a difference of sets.

Difference of Sets **(1.7)**

$$A - B = \{x \mid x \in A \text{ and } x \notin B\}.$$

For example, if $A = \{a, b, c\}$ and $B = \{c, d\}$, then $A - B = \{a, b\}$. We can picture the difference $A - B$ of two general sets A and B by the shaded region of the Venn diagram in Figure 1.6.

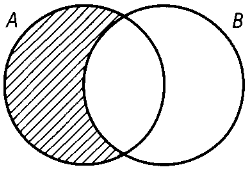


Figure 1.6 Venn diagram of $A - B$.

A natural extension of the difference $A - B$ is the *symmetric difference* of sets A and B, which is the union of $A - B$ with $B - A$ and is denoted by $A \oplus B$. The set $A \oplus B$ is represented by the shaded regions of the Venn diagram in Figure 1.7.

We can define the symmetric difference by using the "exclusive" form of "or" as follows:

Symmetric Difference of Sets **(1.8)**

$$A \oplus B = \{x \mid x \in A \text{ or } x \in B \text{ but not both}\}.$$

As is usually the case, there are many relationships to discover. For example, it's easy to see that

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

Can you verify that $(A \oplus B) \oplus C = A \oplus (B \oplus C)$? For example, try to draw two Venn diagrams, one for each side of the equation.

Complement of a Set

If the discussion always refers to sets that are subsets of a particular set U, then U is called the *universe of discourse*, and the difference $U - A$ is called the *complement* of A, which we denote by A'. The Venn diagram in Figure 1.8 pictures the universe U as a rectangle, with two subsets A and B, where the shaded region represents the complement $(A \cup B)'$.

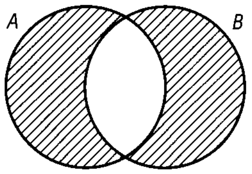


Figure 1.7 Venn diagram of $A \oplus B$.

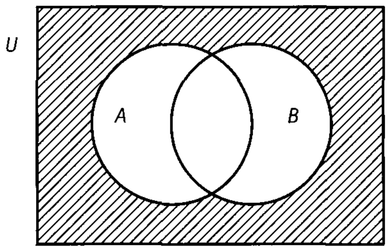


Figure 1.8 Venn diagram of $(A \cup B)'$.

Combining Set Operations

There are many useful properties that combine different set operations. Venn diagrams are often quite useful in trying to visualize sets that are constructed with different operations. For example, the set $A \cap (B \cup C)$ is represented by the shaded regions of the Venn diagram in Figure 1.9.

Here are two distributive properties and two absorption properties that combine the operations of union and intersection.

Combining Properties of Union and Intersection **(1.9)**

a.	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.	($\cap$ distributes over $\cup$.)
b.	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.	($\cup$ distributes over $\cap$.)
c.	$A \cap (A \cup B) = A$.	(absorption law)
d.	$A \cup (A \cap B) = A$.	(absorption law)

example **1.15 A Distributive Proof**

We'll prove the following statement (1.9a) about distribution:

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C).$$

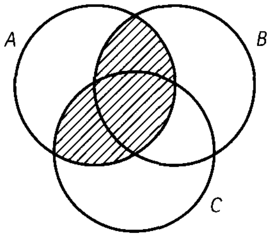


Figure 1.9 Venn diagram of $A \cap (B \cup C)$.

Proof: We'll show that $x \in A \cap (B \cup C)$ if and only if $x \in (A \cap B) \cup (A \cap C)$.

$$\begin{aligned} x \in A \cap (B \cup C) &\text{ iff } x \in A \text{ and } x \in B \cup C \\ &\text{ iff } x \in A, \text{ and either } x \in B \text{ or } x \in C \\ &\text{ iff either } (x \in A \text{ and } x \in B) \text{ or } (x \in A \text{ and } x \in C) \\ &\text{ iff } x \in (A \cap B) \cup (A \cap C). \quad \text{QED.} \end{aligned}$$

end example

The complement operation combines with the other operations in many interesting ways. Here's a list of some of the useful properties.

Properties of Complement **(1.10)**

a. $(A')' = A$.

b. $\varnothing' = U$ and $U' = \varnothing$.

c. $A \cap A' = \varnothing$ and $A \cup A' = U$.

d. $A \subset B$ if and only if $B' \subset A'$.

e. $(A \cup B)' = A' \cap B'$ (De Morgan's law).

f. $(A \cap B)' = A' \cup B'$ (De Morgan's law).

g. $A \cap (A' \cup B) = A \cap B$ (absorption law).

h. $A \cup (A' \cap B) = A \cup B$ (absorption law).

example

1.16 Subset Conditions

We'll prove the following statement (1.10d):

$$A \subset B \text{ if and only if } B' \subset A'.$$

Proof: In this case we're able to connect the two sides of the iff statement with a sequence of iff statements. Be sure that you know the reason for each step.

$$\begin{aligned} A \subset B &\text{ iff } x \in A \text{ implies } x \in B \\ &\text{ iff } x \notin B \text{ implies } x \notin A \\ &\text{ iff } x \in B' \text{ implies } x \in A' \\ &\text{ iff } B' \subset A'. \quad \text{QED.} \end{aligned}$$

end example

1.2.3 Counting Finite Sets

Let's apply some of our knowledge about sets to counting finite sets. The size of a set S is called its *cardinality*, which we'll denote by

$$|S|.$$

For example, if $S = \{a, b, c\}$, then $|S| = |\{a, b, c\}| = 3$. We can say "the cardinality of S is 3," or "3 is the cardinal number of S," or simply "S has three elements."

Counting by Inclusion and Exclusion

Suppose we want to count the union of two sets. For example, suppose we have $A = \{1, 2, 3, 4, 5\}$ and $B = \{2, 4, 6, 8\}$. Since $A \cup B = \{1, 2, 3, 4, 5, 6, 8\}$, it follows that $|A \cup B| = 7$. Similarly, since $A \cap B = \{2, 4\}$, it follows that $|A \cap B| = 2$. If we know any three of the four numbers $|A|$, $|B|$, $|A \cap B|$, and $|A \cup B|$, then we can find the fourth by using the following counting rule for finite sets:

Union Rule **(1.11)**

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

It's easy to discover this rule by drawing a Venn diagram.

The union rule extends to three or more sets. For example, the following calculation gives the union rule for three finite sets:

$$\begin{aligned} |A \cup B \cup C| &= |A \cup (B \cup C)| \qquad (1.12)\\ &= |A| + |B \cup C| - |A \cap (B \cup C)| \\ &= |A| + |B| + |C| - |B \cap C| - |A \cap (B \cup C)| \\ &= |A| + |B| + |C| - |B \cap C| - |(A \cap B) \cup (A \cap C)| \\ &= |A| + |B| + |C| - |B \cap C| - |A \cap B| - |A \cap C| + |A \cap B \cap C|. \end{aligned}$$

The popular name for the union rule and its extensions to three or more sets is the *principle of inclusion and exclusion*. The name is appropriate because the rule says to add (include) the count of each individual set. Then subtract (exclude) the count of all intersecting pairs of sets. Next, include the count of all intersections of three sets. Then exclude the count of all intersections of four sets, and so on.

example 1.17 A Building Project

Suppose A, B, and C are sets of tools needed by three workers on a job. For convenience let's call the workers A, B, and C. Suppose further that the workers share some of the tools (for example, on a housing project, all three workers might share a single table saw). Suppose that A uses 8 tools, B uses 10 tools, and C uses 5 tools. Suppose further that A and B share 3 tools, A and C share 2 tools, and B and C share 2 tools. Finally, suppose that A, B, and C share the use of 2 tools. How many distinct tools are necessary to do the job? Thus we want to find the value

$$|A \cup B \cup C|.$$

We can apply the result of (1.12) to obtain:

$$\begin{aligned}|A \cup B \cup C| &= |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C| \\ &= 8 + 10 + 5 - 3 - 2 - 2 + 2 \\ &= 18 \text{ tools.}\end{aligned}$$

end example

example **1.18 Surveys**

Suppose we survey 200 students to see whether they are taking courses in computer science, mathematics, or physics. The results show that 90 students take computer science, 110 take mathematics, and 60 take physics. Further, 20 students take computer science and mathematics, 20 take computer science and physics, and 30 take mathematics and physics. We are interested in those students that take courses in all three areas. Do we have enough information? Let C, M, and P stand for the sets of students that take computer science, mathematics, and physics. So we are interested in the number

$$|C \cap M \cap P|.$$

From the information given we know that $200 \geq |C \cup M \cup P|$. The reason for the inequality is that some students may not be taking any courses in the three areas. The statistics also give us the following information.

$$\begin{aligned}|C| &= 90 \\ |M| &= 110 \\ |P| &= 60 \\ |C \cap M| &= 20 \\ |C \cap P| &= 20 \\ |M \cap P| &= 30.\end{aligned}$$

Applying the result of (1.12), we have

$$\begin{aligned}200 &\geq |C| + |M| + |P| - |C \cap M| - |C \cap P| - |M \cap P| + |C \cap M \cap P| \\ &= 90 + 110 + 60 - 20 - 20 - 30 + |C \cap M \cap P| \\ &= 190 + |C \cap M \cap P|.\end{aligned}$$

Therefore, $|C \cap M \cap P| \leq 10$. So we can say that at most 10 of the students polled take courses in all three areas. We would get an exact answer if we knew that each student in the survey took at least one course from one of the three areas.

end example

Counting the Difference of Two Sets

Suppose we need to count the difference of two sets. For example, if $A = \{1, 3, 5, 7, 9\}$ and $B = \{2, 3, 4, 8, 10\}$, then $A - B = \{1, 5, 7, 9\}$. So $|A - B| = 4$. If we know any two of the numbers $|A|$, $|A - B|$, and $|A \cap B|$, then we can find the third by using the following counting rule for finite sets.

Difference Rule **(1.13)**

$$|A - B| = |A| - |A \cap B|.$$

It's easy to discover this rule by drawing a Venn diagram. Two special cases of (1.13) are also intuitive and can be stated as follows:

$$\text{If } B \subset A, \text{ then } |A - B| = |A| - |B|. \tag{1.14}$$

$$\text{If } A \cap B = \varnothing, \text{ then } |A - B| = |A|. \tag{1.15}$$

example **1.19 Tool Boxes**

Continuing with the data from Example 1.17, suppose we want to know how many personal tools each worker needs (tools not shared with other workers). For example, Worker A needs a tool box of size

$$|A - (B \cup C)|.$$

We can compute this value using both the difference rule (1.13) and the union rule (1.11) as follows.

$$\begin{aligned}|A - (B \cup C)| &= |A| - |A \cap (B \cup C)| \\ &= |A| - |(A \cap B) \cup (A \cap C)| \\ &= |A| - (|(A \cap B)| + |(A \cap C)| - |A \cap B \cap C|) \\ &= 8 - (3 + 2 - 2) \\ &= 5 \text{ personal tools for A.}\end{aligned}$$

end example

1.2.4 Bags (Multisets)

A *bag* (or *multiset*) is a collection of objects that may contain repeated occurrences of elements. Here are the important characteristics.

Two Characteristics of Bags

1. There may be repeated occurrences of elements.
2. There is no particular order or arrangement of the elements.

To differentiate bags from sets, we'll use brackets to enclose the elements. For example, $[h, u, g, h]$ is a bag with four elements. Two bags A and B are *equal* if the number of occurrences of each element in A or B is the same in either bag. If A and B are equal bags, we write $A = B$. For example, $[h, u, g, h] = [h, h, g, u]$, but $[h, u, g, h] \neq [h, u, g]$.

We can also define the subbag notion. Define A to be a *subbag* of B, and write $A \subset B$, if the number of occurrences of each element x in A is less than or equal to the number of occurrences of x in B. For example, $[a, b] \subset [a, b, a]$, but $[a, b, a] \not\subset [a, b]$. It follows from the definition of subbag that two bags A and B are equal if and only if A is a subbag of B and B is a subbag of A.

If A and B are bags, we define the *sum* of A and B, denoted by $A + B$, as follows: If x occurs m times in A and n times in B, then x occurs $m + n$ times in $A + B$. For example,

$$[2, 2, 3] + [2, 3, 3, 4] = [2, 2, 2, 3, 3, 3, 4].$$

We can define union and intersection for bags also (we will use the same symbols as for sets). Let A and B be bags, and let m and n be the number of times x occurs in A and B, respectively. Put the larger of m and n occurrences of x in $A \cup B$. Put the smaller of m and n occurrences of x in $A \cap B$. For example, we have

$$[2, 2, 3] \cup [2, 3, 3, 4] = [2, 2, 3, 3, 4]$$

and

$$[2, 2, 3] \cap [2, 3, 3, 4] = [2, 3].$$

example 1.20 Least and Greatest

Let $p(x)$ denote the bag of prime numbers that occur in the prime factorization of the natural number x. For example, we have

$$p(54) = [2, 3, 3, 3] \text{ and } p(12) = [2, 2, 3].$$

Let's compute the union and intersection of these two bags. The union gives $p(54) \cup p(12) = [2, 2, 3, 3, 3] = p(108)$, and 108 is the least common multiple of 54 and 12 (i.e., the smallest positive integer that they both divide). Similarly, we get $p(54) \cap p(12) = [2, 3] = p(6)$, and 6 is the greatest common divisor of 54 and 12 (i.e., the largest positive integer that divides them both). Can we discover anything here? It appears that $p(x) \cup p(y)$ and $p(x) \cap p(y)$ compute the least common multiple and the greatest common divisor of x and y. Can you convince yourself?

end example

1.2.5 Sets Should Not Be Too Complicated

Set theory was created by the mathematician Georg Cantor (1845–1918) during the period 1874 to 1895. Later some contradictions were found in the theory. Everything works fine as long as we don't allow sets to be too complicated. Basically, we never allow a set to be defined by a test that checks whether a set is a member of itself. If we allowed such a thing, then we could not decide some questions of set membership. For example, suppose we define the set T as follows:

$$T = \{A \mid A \text{ is a set and } A \notin A\}.$$

In other words, T is the set of all sets that are not members of themselves. Now ask the question "Is $T \in T$?" If so, then the condition for membership in T must hold. But this says that $T \notin T$. On the other hand, if we assume that $T \notin T$, then we must conclude that $T \in T$. In either case we get a contradiction. This example is known as *Russell's paradox*—after the philosopher and mathematician Bertrand Russell (1872–1970).

This kind of paradox led to a more careful study of the foundations of set theory. For example, Whitehead and Russell [1910] developed a theory of sets based on a hierarchy of levels that they called *types*. The lowest type contains individual elements. Any other type contains only sets whose elements are from the next lower type in the hierarchy. We can list the hierarchy of types as T_0, $T_1, \ldots, T_k, \ldots$, where T_0 is the lowest type containing individual elements and in general T_{k+1} is the type consisting of sets whose elements are from T_k. So any set in this theory belongs to exactly one type T_k for some $k \geq 1$.

As a consequence of the definition, we can say that $A \notin A$ for all sets A in the theory. To see this, suppose A is a set of type T_{k+1}. This means that the elements of A are of type T_k. If we assume that $A \in A$, we would have to conclude that A is also a set of type T_k. This says that A belongs to the two types T_k and T_{k+1}, contrary to the fact that A must belong to exactly one type.

Let's examine why Russell's paradox can't happen in this new theory of sets. Since $A \notin A$ for all sets A in the theory, the original definition of T can be simplified to $T = \{A \mid A \text{ is a set}\}$. This says that T contains all sets. But

T itself isn't even a set in the theory because it contains sets of different types. In order for T to be a set in the theory, each A in T must belong to the same type. For example, we could pick some type T_k and define $T = \{A \mid A \text{ has type } T_k\}$. This says that T is a set of type T_{k+1}. Now since T is a set in the theory, we know that $T \notin T$. But this fact doesn't lead us to any kind of contradictory statement.

Exercises

Describing Sets

1. The set $\{x \mid x \text{ is a vowel}\}$ can also be described by $\{a, e, i, o, u\}$. Describe each of the following sets by listing its elements.

   a. $\{x \mid x \in \mathbb{N} \text{ and } 0 < x < 8\}$.
   b. $\{2k + 1 \mid k \text{ is an even integer between 1 and 10}\}$.
   c. $\{x \mid x \text{ is an odd prime less than 20}\}$.
   d. $\{x \mid x \text{ is a month ending with the letter "}y\text{"}\}$.
   e. $\{x \mid x \text{ is a letter in the words MISSISSIPPI RIVER}\}$.
   f. $\{x \mid x \in \mathbb{N} \text{ and } x \text{ divides } 24\}$.

2. The set $\{a, e, i, o, u\}$ can also be described by $\{x \mid x \text{ is a vowel}\}$. Describe each of the following sets in terms of a property of its elements.

   a. The set of dates in the month of January.
   b. $\{1, 3, 5, 7, 9, 11, 13, 15\}$.
   c. $\{1, 4, 9, 16, 25, 36, 49, 64\}$.
   d. The set of even integers $\{..., -4, -2, 0, 2, 4, ...\}$.

Subsets

3. Let $A = \{a, \varnothing\}$. Answer true or false for each of the following statements.

   a. $a \in A$. b. $\{a\} \in A$. c. $a \subset A$. d. $\{a\} \subset A$.
   e. $\varnothing \subset A$. f. $\varnothing \in A$. g. $\{\varnothing\} \subset A$. h. $\{\varnothing\} \in A$.

4. Show that $\varnothing \subset A$ for every set A.

5. Find two finite sets A and B such that $A \in B$ and $A \subset B$.

6. Write down the power set for each of the following sets.

   a. $\{x, y, z, w\}$. b. $\{a, \{a, b\}\}$. c. $\varnothing$.
   d. $\{\varnothing\}$. e. $\{\{a\}, \varnothing\}$.

7. For each collection of sets, find the smallest set A such that the collection is a subset of power(A).

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

Set Operations

8. Suppose A and B are sets defined as follows:

$$A = \{x \mid x = 4k + 1 \text{ and } k \in \mathbb{N}\},$$
$$B = \{x \mid x = 3k + 5 \text{ and } k \in \mathbb{N}\}.$$

   a. List ten elements of $A \cup B$.
   b. List four elements of $A \cap B$.

9. Is power$(A \cup B)$ = power$(A) \cup$ power(B)?

10. For each integer i, define A_i as follows:
   If i is even then $A_i = \{x \mid x \in \mathbb{Z} \text{ and } x < -i \text{ or } i < x\}$.
   If i is odd then $A_i = \{x \mid x \in \mathbb{Z} \text{ and } -i < x < i\}$.

   a. Describe each of the sets A_0, A_1, A_2, A_3, A_{-2}, and A_{-3}.
   b. Find the union of the collection $\{A_i \mid i \in \{1, 3, 5, 7, 9\}\}$.
   c. Find the union of the collection $\{A_i \mid i \text{ is even}\}$.
   d. Find the union of the collection $\{A_i \mid i \text{ is odd}\}$.
   e. Find the union of the collection $\{A_i \mid i \in \mathbb{N}\}$.
   f. Find the intersection of the collection $\{A_i \mid i \in \{1, 3, 5, 7, 9\}\}$.
   g. Find the intersection of the collection $\{A_i \mid i \text{ is even}\}$.
   h. Find the intersection of the collection $\{A_i \mid i \text{ is odd}\}$.
   i. Find the intersection of the collection $\{A_i \mid i \in \mathbb{N}\}$.

11. For each natural number n, let A_n be defined by

$$A_n = \{x \mid x \in \mathbb{N} \text{ and } x \text{ divides } n \text{ with no remainder}\}.$$

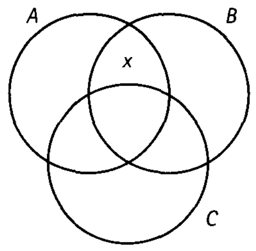
   a. Describe each of the sets A_0, A_1, A_2, A_3, A_4, A_5, A_6, A_7, and A_{100}.
   b. Find the union of the collection $\{A_n \mid n \in \{1, 2, 3, 4, 5, 6, 7\}\}$.
   c. Find the intersection of the collection $\{A_n \mid n \in \{1, 2, 3, 4, 5, 6, 7\}\}$.
   d. Find the union of the collection $\{A_n \mid n \in \mathbb{N}\}$.
   e. Find the intersection of the collection $\{A_n \mid n \in \mathbb{N}\}$.

12. For each of the following expressions, use a Venn diagram representing a universe U and two subsets A and B. Shade the part of the diagram that corresponds to the given set.

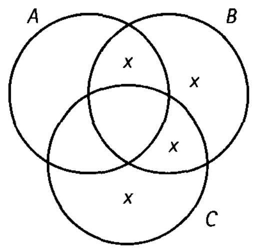
   a. A'. b. B'. c. $(A \cup B)'$.
   d. $A' \cap B'$. e. $A' \cup B'$. f. $(A \cap B)'$.

13. Each Venn diagram in the following figure represents a set whose regions are indicated by the letter x. Find an expression for each of the three sets in terms of set operations. Try to simplify your answers.

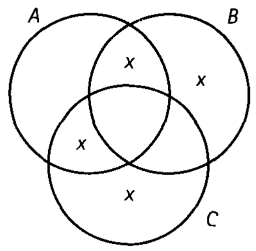
a.



b.



c.



Counting Finite Sets

14. Discover an inclusion exclusion formula for the number of elements in the union of four sets A, B, C, and D.

15. Given three sets A, B, and C. Suppose the union of the three sets has cardinality 280. Suppose also that $|A| = 100$, $|B| = 200$, and $|C| = 150$. And suppose we also know $|A \cap B| = 50$, $|A \cap C| = 80$, and $|B \cap C| = 90$. Find the cardinality of the intersection of the three given sets.

16. Suppose A, B, and C represent three bus routes through a suburb of your favorite city. Let A, B, and C also be sets whose elements are the bus stops for the corresponding bus route. Suppose A has 25 stops, B has 30 stops, and C has 40 stops. Suppose further that A and B share (have in common) 6 stops, A and C share 5 stops, and B and C share 4 stops. Lastly, suppose that A, B, and C share 2 stops. Answer each of the following questions.

    a. How many distinct stops are on the three bus routes?
    b. How many stops for A are not stops for B?
    c. How many stops for A are not stops for both B and C?
    d. How many stops for A are not stops for any other bus?

17. Suppose a highway survey crew noticed the following information about 500 vehicles: In 100 vehicles the driver was smoking, in 200 vehicles the driver was talking to a passenger, and in 300 vehicles the driver was tuning the radio. Further, in 50 vehicles the driver was smoking and talking, in 40 vehicles the driver was smoking and tuning the radio, and in 30 vehicles the driver was talking and tuning the radio. What can you say about the number of drivers who were smoking, talking, and tuning the radio?

18. Suppose the following people went to a summer camp: 27 boys, 15 city children, 27 men, 21 noncity boys, 42 people from the city, 18 city males, and 21 noncity females. How many people went to summer camp?

Bags

19. Find the union and intersection of each of the following pairs of bags.

    a. $[x, y]$ and $[x, y, z]$.
    b. $[x, y, x]$ and $[y, x, y, x]$.
    c. $[a, a, a, b]$ and $[a, a, b, b, c]$.
    d. $[1, 2, 2, 3, 3, 4, 4]$ and $[2, 3, 3, 4, 5]$.
    e. $[x, x, [a, a], [a, a]]$ and $[a, a, x, x]$.
    f. $[a, a, [b, b], [a, [b]]]$ and $[a, a, [b], [b]]$.

20. Find a bag B that solves the following two simultaneous bag equations:

$$B \cup [2,2,3,4] = [2,2,3,3,4,4,5],$$
$$B \cap [2,2,3,4,5] = [2,3,4,5].$$

21. How would you define the difference operation for bags? Try to make your definition agree with the difference operation for sets whenever the bags are like sets (without repeated occurrences of elements).

Proofs and Challenges

22. Prove each of the following facts about the union operation (1.3). Use subset arguments that are written in complete sentences.

    a. $A \cup \varnothing = A$.
    b. $A \cup B = B \cup A$.
    c. $A \cup A = A$.
    d. $A \cup (B \cup C) = (A \cup B) \cup C$.

23. Prove each of the following facts about the intersection operation (1.6). Use subset arguments that are written in complete sentences.

    a. $A \cap \varnothing = \varnothing$.
    b. $A \cap B = B \cap A$.
    c. $A \cap (B \cap C) = (A \cap B) \cap C$.
    d. $A \cap A = A$.
    e. $A \subset B$ if and only if $A \cap B = A$.

24. Prove that power$(A \cap B)$ = power$(A) \cap$ power(B).

25. Prove that $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.

26. Prove each of the following absorption laws (1.9) twice. The first proof should use subset arguments. The second proof should use an already known result.

    a. $A \cap (B \cup A) = A$.
    b. $A \cup (B \cap A) = A$.

27. Show that $(A \cap B) \cup C = A \cap (B \cup C)$ if and only if $C \subset A$.

28. Give a proof or a counterexample for each of the following statements.
    a. $A \cap (B \cup \mathrm{A}) = A \cap B$.
    b. $A - (B \cap \mathrm{A}) = A - B$.
    c. $A \cap (B \cup C) = (A \cup B) \cap (A \cup C)$.
    d. $A \oplus A = A$.

29. Prove each of the following properties of the complement (1.10).

a.	$(A')' = A$.	
b.	$\varnothing' = U$ and $U' = \varnothing$.	
c.	$A \cap A' = \varnothing$ and $A \cup A' = U$.	
d.	$(A \cup B)' = A' \cap B'$.	(De Morgan's law)
e.	$(A \cap B)' = A' \cup B'$.	(De Morgan's law)
f.	$A \cap (A' \cup B) = A \cap B$.	(absorption law)
g.	$A \cup (A' \cap B) = A \cup B$.	(absorption law)

30. Try to find a description of a set A satisfying the equation $A = \{a, A, b\}$. Notice in this case that $A \in A$.

1.3 Ordered Structures

In the previous section we saw that sets and bags are used to represent unordered information. In this section we'll introduce some notions and notations for structures that have some kind of ordering to them.

1.3.1 Tuples

When we write down a sentence, it always has a sequential nature. For example, in the previous sentence the word "When" is the first word, the word "we" is the second word, and so on. Informally, a *tuple* is a collection of things, called its *elements,* where there is a first element, a second element, and so on. The elements of a tuple are also called *members*, *objects*, or *components.* We'll denote a tuple by writing down its elements, separated by commas, and surrounding everything with the two symbols "(" and ")". For example, the tuple (12, R, 9) has three elements. The first element is 12, the second element is the letter R, and third element is 9. The beginning sentence of this paragraph can be represented by the following tuple:

(When, we, write, down, . . . , sequential, nature).

If a tuple has n elements, we say that its *length* is n, and we call it an *n-tuple.* So the tuple $(8, k, \text{hello})$ is a 3-tuple, and $(x_1, \ldots, x_8)$ is an 8-tuple. The 0-tuple is denoted by $(\ )$, and we call it the *empty* tuple. A 2-tuple is often called an *ordered pair*, and a 3-tuple might be called an *ordered triple.* Other words used in place of the word *tuple* are *vector* and *sequence*, possibly modified by the word *ordered.*

Two n-tuples $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ are said to be *equal* if $x_i = y_i$ for $1 \leq i \leq n$, and we denote this by $(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$. Thus the ordered pairs $(3, 7)$ and $(7, 3)$ are not equal, and we write $(3, 7) \neq (7, 3)$. Since tuples convey the idea of order, they are different from sets and bags. Here are some examples:

Sets:	$\{b, a, t\} = \{t, a, b\}$.
Bags:	$[t, o, o, t] = [o, t, t, o]$.
Tuples:	$(t, o, o, t) \neq (o, t, t, o)$ and $(b, a, t) \neq (t, a, b)$.

Here are the two important characteristics of tuples.

Two Characteristics of Tuples

1. There may be repeated occurrences of elements.
2. There is an order or arrangement of the elements.

The rest of this section introduces structures that are represented as tuples. We'll also see in the next section that graphs and trees are often represented using tuples.

Cartesian Product of Sets

We often need to represent information in the form of tuples, where the elements in each tuple come from known sets. Such a set is called a Cartesian product, in honor of René Descartes (1596–1650), who introduced the idea of graphing ordered pairs. The Cartesian product is also referred to as the *cross product.* Here's the formal definition.

Definition of Cartesian Product

If A and B are sets, then the *Cartesian product* of A and B, which is denoted by $A \times B$, is the set of all ordered pairs (a, b) such that $a \in A$ and $b \in B$. In other words, we have

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

For example, if $A = \{x, y\}$ and $B = \{0, 1\}$, then

$$A \times B = \{(x, 0), (x, 1), (y, 0), (y, 1)\}.$$

Suppose we let $A = \varnothing$ and $B = \{0, 1\}$ and then ask the question: "What is $A \times B$?". If we apply the definition of Cartesian product, we must conclude that there are no ordered pairs with first elements from the empty set. Therefore, $A \times B = \varnothing$. So it's easy to generalize and say that $A \times B$ is nonempty if and only if both A and B are nonempty sets. The Cartesian product of two sets is easily extended to any number of sets $A_1, \ldots, A_n$ by writing

$$A_1 \times \cdots \times A_n = \{(x_1, \ldots, x_n) \mid x_i \in A_i\}.$$

If all the sets A_i in a Cartesian product are the same set A, then we use the abbreviated notation $A^n = A \times \cdots \times A$. With this notation we have the following definitions for the sets A^1 and A^0:

$$A^1 = \{(a) \mid a \in A\} \text{ and } A^0 = \{(\ )\}.$$

So we must conclude that $A^1 \neq A$ and $A^0 \neq \varnothing$.

example

1.21 Some Products

Let $A = \{a, b, c\}$. Then we have the following Cartesian products:

$$\begin{aligned}
&A^0 = \{()\},\\
&A^1 = \{(a), (b), (c)\},\\
&A^2 = \{(a, a), (a, b), (a, c), (b, a), (b, b), (b, c), (c, a), (c, b), (c, c)\},
\end{aligned}$$

A^3 is bigger yet, with twenty-seven 3-tuples.

end example

When working with tuples, we need the ability to randomly access any component. The components of an n-tuple can be indexed in several different ways depending on the problem at hand. For example, if $t \in A \times B \times C$, then we might represent t in any of the following ways:

$$\begin{aligned}
&(t_1, t_2, t_3),\\
&(t(1), t(2), t(3)),\\
&(t[1], t[2], t[3]),\\
&(t(A), t(B), t(C)),\\
&(A(t), B(t), C(t)).
\end{aligned}$$

Let's look at an example that shows how Cartesian products and tuples are related to some familiar objects of programming.

example

1.22 Arrays, Matrices, and Records

In computer science, a 1-dimensional array of size n with elements in the set A is an n-tuple in the Cartesian product A^n. So we can think of the Cartesian product A^n as the set of all 1-dimensional arrays of size n over A. If $x = (x_1, \ldots, x_n)$, then the component x_i is usually denoted—in programming languages—by $x[i]$.

A 2-dimensional array—also called a *matrix*—can be thought of as a table of objects that are indexed by rows and columns. If x is a matrix with m rows and n columns, we say that x is an m by n matrix. For example, if x is a 3 by 4 matrix, then x can be represented by the following diagram:

$$x = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \end{bmatrix}.$$

We can also represent x as a 3-tuple whose components are 4-tuples as follows:

$$x = ((x_{11}, x_{12}, x_{13}, x_{14}), (x_{21}, x_{22}, x_{23}, x_{24}), (x_{31}, x_{32}, x_{33}, x_{34})).$$

In programming, the component x_{ij} is usually denoted by $x[i, j]$. We can think of the Cartesian product $(A^4)^3$ as the set of all 2-dimensional arrays over A with 3 rows and 4 columns. Of course, this idea extends to higher dimensions. For example, $((A^5)^7)^4$ represents the set of all 3-dimensional arrays over A consisting of 4-tuples whose components are 7-tuples whose components are 5-tuples of elements of A.

For another example, we can think of $A \times B$ as the set of all records, or structures, with two fields A and B. For a record $r = (a, b) \in A \times B$ the components a and b are normally denoted by $r.A$ and $r.B$.

end example

There are at least three nice things about tuples: They are easy to understand; they are basic building blocks for the representation of information; and they are easily implemented by a computer, which we'll discuss in the next example.

example

1.23 Computer Repesentation of Tuples

Computers represent tuples in contiguous cells of memory so that each component can be accessed quickly. For example, suppose that each component of the tuple $x = (x_1, \ldots, x_n)$ needs M memory cells to store it. If B is the beginning address of memory allocated for the tuple x, then x_1 is at location B, x_2 is at location $B + M$, and in general, x_k is at location

$$B + M(k - 1).$$

So each component x_k of x can be accessed in the amount of time that it takes to evaluate $B + M(k - 1)$.

For multidimensional arrays, the access time is also fast. For example, suppose x is a 3 by 4 matrix represented in the following "row-major" form as a three tuple of rows, where each row is a 4-tuple.

$$x = ((x_{11}, x_{12}, x_{13}, x_{14}), (x_{21}, x_{22}, x_{23}, x_{24}), (x_{31}, x_{32}, x_{33}, x_{34})).$$

Suppose that each component of x needs M memory cells. If B is the beginning address of memory allocated for x, then x_{11} is at location B, x_{21} is at location $B + 4M$, and x_{31} is at location $B + 8M$. The location of an arbitrary element x_{jk} is given by the expression

$$B + 4M(j - 1) + M(k - 1).$$

Expressions such as this are called *address polynomials.* Each component x_{jk} can be accessed in the amount of time that it takes to evaluate the address polynomial, which is close to a constant for any j and k.

end example

1.3.2 Lists

A *list* is a finite ordered sequence of zero or more elements that can be repeated. At this point a list seems just like a tuple. So what's the difference between tuples and lists? The difference—a big one in computer science—is in what parts can be randomly accessed. In the case of tuples we can randomly access any component in a constant amount of time. In the case of lists we can randomly access only two things in a constant amount of time: the first component of a list, which is called its *head,* and the list made up of everything except the first component, which is called its *tail.*

So we'll use a different notation for lists. We'll denote a list by writing down its elements, separated by commas, and surrounding everything with the two symbols "$\langle$ " and "$\rangle$". The *empty list* is denoted by

$$\langle\ \rangle.$$

The number of elements in a list is called its *length.* For example, the list

$$\langle w, x, y, z \rangle,$$

has length 4, its head is w, and its tail is the list $\langle x, y, z \rangle$. If L is a list, we'll use the notation

$$\text{head}(L) \text{ and } \text{tail}(L)$$

to denote the head of L and the tail of L. For example,

$$\text{head}(\langle w, x, y, z \rangle) = w,$$
$$\text{tail}(\langle w, x, y, z \rangle) = \langle x, y, z \rangle.$$

Notice that the empty list $\langle\rangle$ does not have a head or tail.

An important computational property of lists is the ability to easily construct a new list by adding a new element at the head of an existing list. The name *cons* will be used to denote this construction operation. If h is an element of some kind and L is a list, then

$$\text{cons}(h,\ L)$$

denotes the list whose head is h and whose tail is L. Here are a few examples:

$$\begin{aligned}&\text{cons}(w,\ \langle x,\ y,\ z\rangle) = \langle w,\ x,\ y,\ z\rangle,\\ &\text{cons}(a,\ \langle\ \rangle) = \langle a\rangle.\\ &\text{cons}(\text{this},\ \langle \text{is, helpful}\rangle) = \langle \text{this, is, helpful}\rangle.\end{aligned}$$

The operations head, tail, and cons can be done efficiently and dynamically during the execution of a program. The three operations are related by the following equation for any nonempty list L.

$$\text{cons}(\text{head}(L),\ \text{tail}(L)) = L.$$

There is no restriction on the kind object that a list can contain. In fact, it is often quite useful to represent information in the form of lists whose elements may be lists, and the elements of those lists may be lists, and so on. Here are a few examples of such lists, together with their heads and tails.

L	head(L)	tail(L)
$\langle a, \langle b\rangle\rangle$	a	$\langle\langle b\rangle\rangle$
$\langle\langle a\rangle, \langle b, a\rangle\rangle$	$\langle a\rangle$	$\langle\langle b, a\rangle\rangle$
$\langle\langle\langle\ \rangle, a, \langle\ \rangle\rangle, b, \langle\ \rangle\rangle$	$\langle\langle\ \rangle, a, \langle\ \rangle\rangle$	$\langle b, \langle\ \rangle\rangle$

If all the elements of a list L are from a particular set A, then L is said to be a *list over* A. For example, each of the following lists is a list over $\{a,\ b,\ c\}$.

$$\langle\ \rangle,\ \langle a\rangle,\ \langle a,\ b\rangle,\ \langle b,\ a\rangle,\ \langle b,\ c,\ a,\ b,\ c\rangle.$$

We'll denote the collection of all lists over A by

$$\text{lists}(A).$$

There are at least four nice things about lists: They are easy to understand; they are basic building blocks for the representation of information; they are easily implemented by a computer; and they are easily manipulated by a computer program. We'll discuss this in the next example.

example 1.24 Computer Repesentation of Lists

A simple way to represent a list in a computer is to allocate a block of memory for each element of the list. The block of memory contains the element together

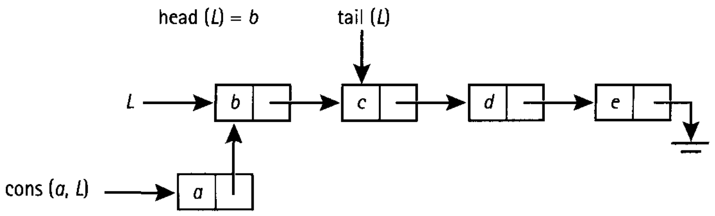


Figure 1.10 Memory representation of a list.

with an address (called a pointer or link) to the next block of memory for the next element of the list. In this way there is no need to have list elements next to each other in memory, so that the creation and deletion of list elements can occur dynamically during the execution of a program.

For example, let's consider the list $L = \langle b, c, d, e \rangle$. Figure 1.10 shows the memory representation of L in which each arrow represents an address (i.e., a pointer or link) and each box represents a block of memory containing an element of the list and an arrow pointing to the address of the next box. The last arrow in the box for e points to the "ground" symbol to signify the end of the list. Empty lists point to the ground symbol, too. The figure also shows head$(L) = b$ and tail$(L) = \langle c, d, e \rangle$. So head and tail are easily calculated from L. The figure also shows how the cons operation constructs a new list cons$(a, L) = \langle a, b, c, d, e \rangle$ by allocating a new block of memory to contain a and a pointer to L.

end example

1.3.3 Strings and Languages

A *string* is a finite ordered sequence of zero or more elements that are placed next to each other in juxtaposition. The individual elements that make up a string are taken from a finite set called an *alphabet*. If A is an alphabet, then a string of elements from A is said to be a string over A. For example, here are a few strings over $\{a, b, c\}$:

$$a, ba, bba, aacabb.$$

The string with no elements is called the *empty string*, and we denote it by the Greek capital letter lambda:

$$\Lambda.$$

The number of elements that occur in a string s is called the *length* of s, which we sometimes denote by

$$|s|.$$

For example, $|\Lambda| = 0$ and $|aacabb| = 6$ over the alphabet $\{a, b, c\}$.

Concatenation of Strings

The operation of placing two strings s and t next to each other to form a new string st is called *concatenation*. For example, if aab and ba are two strings over the alphabet $\{a, b\}$, then the concatenation of aab and ba is the string

$$aabba.$$

We should note that if the empty string occurs as part of another string, then it does not contribute anything new to the string. In other words, if s is a string, then

$$s\Lambda = \Lambda s = s.$$

Strings are used in the world of written communication to represent information: computer programs; written text in all the languages of the world; and formal notation for logic, mathematics, and the sciences.

There is a strong association between strings and lists because both are defined as finite sequences of elements. This association is important in computer science because computer programs must be able to recognize certain kinds of strings. This means that a string must be decomposed into its individual elements, which can then be represented by a list. For example, the string $aacabb$ can be represented by the list $\langle a, a, c, a, b, b\rangle$. Similarly, the empty string Λ can be represented by the empty list $\langle\ \rangle$.

Languages (Sets of Strings)

A *language* is a set of strings. If A is an alphabet, then a *language* over A is a set of strings over A. The set of all strings over A is denoted by A^*. So any language over A is a subset of A^*. Four simple examples of languages over an alphabet A are the sets $\varnothing$, $\{\Lambda\}$, A, and A^*.

For example, if $A = \{a\}$, then these four simple languages over A become

$$\varnothing,\ \{\Lambda\},\ \{a\},\ \text{and}\ \{\Lambda, a, aa, aaa, \ldots\}.$$

For any natural number n the concatenation of a string s with itself n times is denoted by s^n. For example,

$$s^0 = \Lambda,\ s^1 = s,\ s^2 = ss,\ \text{and}\ s^3 = sss.$$

example 1.25 Language Representations

The exponent notation allows us to represent some languages in a nice concise manner. Here are a few examples of languages.

$\{a^n \mid n \in \mathbb{N}\} = \{\Lambda, a, aa, aaa, \ldots\}.$
$\{ab^n \mid n \in \mathbb{N}\} = \{a, ab, abb, abbb, \ldots\}.$

$$\{a^n b^n \mid n \in \mathbb{N}\} = \{\Lambda,\ ab,\ aabb,\ aaabbb,\ \ldots\}$$
$$\{(ab)^n \mid n \in \mathbb{N}\} = \{\Lambda,\ ab,\ abab,\ ababab,\ \ldots\}.$$

end example

example

1.26 Numerals

A *numeral* is a written number. In terms of strings, we can say that a numeral is a nonempty string of symbols that represents a number. Most of us are familiar with the following three numeral systems. The *Roman numerals* represent the set of nonnegative integers by using the alphabet

$$\{\text{I, V, X, L, C, D, M}\}.$$

The *decimal numerals* represent the set of natural numbers by using the alphabet

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

The *binary numerals* represent the natural numbers by using the alphabet

$$\{0, 1\}.$$

For example, the Roman numeral MDCLXVI, the decimal numeral 1666, and the binary numeral 11010000010 all represent the same number.

end example

Products of Languages

Since languages are sets of strings, they can be combined by the usual set operations of union, intersection, difference, and complement. But there is another important way to combine languages.

We can combine two languages L and M to obtain the set of all concatenations of strings in L with strings in M. This new language is called the *product* of L and M and is denoted by LM. Here's a formal definition.

Product of Languages

The product of languages L and M is the language

$$LM = \{st \mid s \in L \text{ and } t \in M\}.$$

For example, if $L = \{ab,\ ac\}$ and $M = \{a,\ bc,\ abc\}$, then the product LM is the language

$$LM = \{aba,\ abbc,\ ababc,\ aca,\ acbc,\ acabc\}.$$

It's easy to see, from the definition of product, that the following simple properties hold for any language L.

$$L\{\Lambda\} = \{\Lambda\}L = L.$$
$$L\varnothing = \varnothing L = \varnothing.$$

It's also easy to see that the product is associative. In other words, if L, M, and N are languages, then $L(MN) = (LM)N$. Thus we can write down products without using parentheses. On the other hand, it's easy to see that the product is not commutative. In other words, we can find two languages L and M such that $LM \neq ML$.

For any natural number n, the product of a language L with itself n times is denoted by L^n. In other words, we have

$$L^n = \{s_1 s_2 \ldots s_n \mid s_k \in L \text{ for each } k\}.$$

The special case when $n = 0$ has the following definition.

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

example **1.27 Some Language Products**

We'll calculate some products for the langauge $L = \{a, bb\}$.

$L^0 = \{\Lambda\}$,
$L^1 = L = \{a, bb\}$,
$L^2 = LL = \{aa, abb, bba, bbbb\}$,
$L^3 = LL^2 = \{aaa, aabb, abba, abbbb, bbaa, bbabb, bbbba, bbbbbb\}$.

end example

Closure of a Language

If L is a language, then the *closure* of L, denoted by L^*, is the set of all possible concatenations of strings from L. In other words, we have

$$L^* = L^0 \cup L^1 \cup L^2 \cup \cdots \cup L^n \cup \cdots.$$

So $x \in L^*$ if and only if $x \in L^n$ for some n. Therefore, we have

$$x \in L^* \text{ if and only if either } x = \Lambda \text{ or } x = l_1 l_2 \ldots l_n$$

for some $n \geq 1$, where $l_k \in L$ for $1 \leq k \leq n$.

If L is a language, then the *positive closure* of L, which is denoted by L^+, is defined by

$$L^+ = L^1 \cup L^2 \cup L^3 \cup \cdots.$$

It follows from the definition that $L^* = L^+ \cup \{\Lambda\}$. But it's not necessarily true that $L^+ = L^* - \{\Lambda\}$. For example, if $L = \{\Lambda, a\}$, then $L^+ = L^*$.

We should observe that any alphabet A is itself a language and its closure coincides with our original definition of A^* as the set of all strings over A. The following properties give some basic facts about the closure of languages.

Properties of Closure **(1.16)**

a. $\{\Lambda\}^* = \varnothing^* = \{\Lambda\}$.

b. $\Lambda \in L$ if and only if $L^+ = L^*$.

c. $L^* = L^*L^* = (L^*)^*$.

d. $(L^*M^*)^* = (L^* \cup M^*)^* = (L \cup M)^*$.

e. $L(ML)^* = (LM)^*L$.

Proof: We'll prove part (e) and leave the others as exercises. We'll start by examining the structure of an arbitrary string $x \in L(ML)^*$. Since $L(ML)^*$ is the product of L and $(ML)^*$, we can write $x = ly$, where $l \in L$ and $y \in (ML)^*$. Since $y \in (ML)^*$, it follows that $y \in (ML)^n$ for some n. If $n = 0$, then $y = \Lambda$ and we have $x = ly = l\Lambda = l \in L$.

If $n > 0$, then $y = w_1 \ldots w_n$, where $w_k \in ML$ for $1 \leq k \leq n$. So we can write each w_k in the form $w_k = m_k l_k$, where $m_k \in M$ and $l_k \in L$. Now we can collect our facts and write x as a concatenation of strings from L and M.

$$\begin{aligned} x &= ly && \text{where } l \in L \text{ and } y \in (ML)^* \\ &= l(w_1 \ldots w_n) && \text{where } l \in L \text{ and each } w_k \in (ML) \\ &= l(m_1 l_1 \ldots m_n l_n) && \text{where } l \in L \text{ and each } l_k \in L, \text{ and } m_k \in M. \end{aligned}$$

Since we can group strings with parentheses any way we want, we can put things back together in the following order.

$$\begin{aligned} &= (l m_1 l_1 \ldots m_n) l_n && \text{where } l \in L \text{ and each } l_k \in L, \text{ and } m_k \in M \\ &= (z_1 \ldots z_n) l_n && \text{where each } z_k \in LM \text{ and } l_n \in L \\ &= u l_n && \text{where } u \in (LM)^* \text{ and } l_n \in L. \end{aligned}$$

So $x \in (LM)^*L$. Therefore, $L(ML)^* \subset (LM)^*L$. The argument is reversible. So we have $L(ML)^* = (LM)^*L$. QED.

example 1.28 Decimal Numerals

The product is a useful tool for describing languages in terms of simpler languages. For example, suppose we need to describe the language L of all strings of the form $a.b$, where a and b are decimal numerals. For example, the strings 0.45, 1.569, 000.34000 are elements of L. If we let

$$D = \{.\} \text{ and } N = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\},$$

Then we can describe L as the following product in terms of D and N.

$$L = N(N)^* \, DN(N)^*.$$

end example

1.3.4 Relations

Definition of Relation

Ideas such as kinship, connection, and association of objects are keys to the concept of a relation. Informally, a *relation* is a set of n-tuples, where the elements in each tuple are related in some way.

For example, the parent-child relation can be described as the following set of ordered pairs:

$$\text{isParentOf} = \{(x, y) \mid x \text{ is a parent of } y\}.$$

For another example, recall from geometry that if the sides of a right triangle have lengths x, y, and z, where z is the hypotenuse, then $x^2 + y^2 = z^2$. Any 3-tuple of positive real numbers (x, y, z) with this propoerty is called a *Pythagorean triple.* For example, $(1, \sqrt{3}, 2)$ and (3, 4, 5) are Pythagorean triples. The Pythagorean triple relation can be described as the following set of ordered triples:

$$PT = \{(x, y, z) \mid x^2 + y^2 = z^2\}.$$

When we discuss relations in terms of where the tuples come from, there is some terminology that can be helpful.

Definition of Relation

If R is a subset of $A_1 \times \cdots \times A_n$, then R is said to be *an n-ary relation on* (or *over*) $A_1 \times \cdots \times A_n$. If R is a subset of A^n, then we say R is *an n-ary relation on A*. Instead of 1-ary, 2-ary, and 3-ary, we say *unary*, *binary*, and *ternary*.

For example, the isParentOf relation is a binary relation on the set of people and the Pythagorean triple relation is a ternary operation on the set of positive real numbers. In formal terms, if P is the set of all people who are living or who have ever lived, then

$$\text{isParentOf} \subset P \times P.$$

Similarly, if we let $\mathbb{R}^+$ denote the set of positive real numbers, then

$$PT \subset \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{R}^+.$$

Since there are many subsets of a set, there can be many relations. The smallest relation is the empty set $\varnothing$, which is called the *empty relation.* The largest relation is $A_1 \times \cdots \times A_n$ itself, which is called the *universal relation.*

If R is a relation and $(x_1, \ldots, x_n) \in R$, this fact is often denoted by the *prefix expression*

$$R(x_1, \ldots, x_n).$$

For example, $PT(1, \sqrt{3}, 2)$ means that $(1, \sqrt{3}, 2) \in PT$.

If R is a binary relation, then the statement $(x, y) \in R$ can be denoted by $R(x, y)$, but it is often denoted by the *infix expression*

$$x\ R\ y.$$

For example, "John isParentOf Mary" means that (John, Mary) $\in$ isParentOf.

We use many binary relations without even thinking about it. For example, we use the less-than relation on numbers m and n by writing $m < n$ instead of $(m, n) \in <$ or $< (m, n)$. We also use equality without thinking about it as a binary relation. The *equality relation* on a set A is the set

$$\{(x, x) \mid x \in A\}.$$

For example, if $A = \{a, b, c\}$, then the equality relation on A is the set $\{(a, a), (b, b), (c, c)\}$. We normally denote equality by the symbol $=$ and we write $a = a$ instead of $(a, a) \in =$ or $= (a, a)$.

Since unary relations are sets of 1-tuples, we usually dispense with the tuple notation and simply write a unary relation as a set of elements. For example, instead of writing $R = \{(2), (3), (5), (7)\}$ we write $R = \{2, 3, 5, 7\}$. So $R(2)$ and $2 \in R$ mean the same thing.

Relational Databases

A *relational database* is a collection of facts that are represented by tuples in such a way that the tuples can be accessed in various ways to answer queries about the facts. To accomplish these tasks each component of a tuple must have an associated name, called an *attribute.*

For example, suppose we have a database called Borders that describes the foreign countries and large bodies of water that border each state of the United States. The following table represents a sample of the information in the database with attribute names State, Foreign, and Water.

Borders

State	Foreign	Water
Washington	Canada	Pacific Ocean
Minnesota	Canada	Lake Superior
Wisconsin	None	Lake Michigan
Oregon	None	Pacific Ocean
Maine	Canada	Atlantic Ocean
Michigan	Canada	Lake Superior
Michigan	Canada	Lake Huron
Michigan	Canada	Lake Michigan
California	Mexico	Pacific Ocean
Arizona	Mexico	None

There is no special order to the rows of a relational database. So the table can be represented as a set of tuples.

$$\begin{aligned}\text{Borders} \;=\; \{ & (\text{Washington, Canada, Pacific Ocean}),\\ & (\text{Minnesota, Canada, Lake Superior}),\\ & (\text{Wisconsin, None, Lake Michigan}), \ldots \}.\end{aligned}$$

example

1.29 Questions about Borders

Let's look at a few questions or queries that can be asked about the Borders database. Each question can be answered by describing a set or a relation. For example, suppose we ask the question

What states border Mexico?

The answer is the set

$$\{x \mid (x, \text{Mexico}, z) \in \text{Borders, for some } z\}.$$

Here are a few more questions that we'll leave for the exercises.

What bodies of water border Michigan?
What states border the Pacific Ocean?
What states are landlocked?
What relation represents the state-water pairs?
What state-water pairs contain a state bordering Canada?

end example

Queries can be answered not only by describing a set or a relation as in the example, but also by describing an expression in terms of basic operations that construct new relations by selecting certain tuples, by eliminating certain attributes, or by combining attributes of two relations. We'll discuss these basic operations on relational databases in Section 10.4.

1.3.5 Counting Tuples

How can we count a set of tuples, lists, or strings? Since tuples, lists, and strings represent finite ordered sequences of objects, the only difference is how we represent them, not whether there are more of one kind than another. For example, over the set $\{a, b\}$ there are eight 3-tuples, eight lists of length 3, and eight strings of length 3. So without any loss of generality we'll discuss counting sets of tuples. The main tools that we'll use are the rules for counting Cartesian products of finite sets.

The Product Rule

Suppose we need to know the cardinality of $A \times B$ for finite sets A and B. In other words, we want to know how many 2-tuples are in $A \times B$. For example, suppose that $A = \{a, b, c\}$ and $B = \{0, 1, 2, 3\}$. The sets A and B are small enough so that we can write down all 12 of the tuples. The exercise might also help us notice that each element of A can be paired with any one of the four elements in B. Since there are three elements in A, it follows that

$$|A \times B| = (3)(4) = 12.$$

This is an example of a general counting technique called the product rule, which we'll state as follows for any two finite sets A and B.

Product Rule **(1.17)**

$$|A \times B| = |A||B|.$$

It's easy to see that (1.17) generalizes to a Cartesian product of three or more finite sets. For example, $A \times B \times C$ and $A \times (B \times C)$ are not actually equal because an arbitrary element in $A \times \mathrm{B} \times C$ is a 3-tuple (a, b, c), while an arbitrary element in $A \times (B \times C)$ is a 2-tuple $(a, (b, c))$. Still the two sets have the same cardinality. Can you convince yourself of this fact? Now proceed as follows:

$$\begin{aligned} |A \times B \times C| &= |A \times (B \times C)| \\ &= |A|\,|B \times C| \\ &= |A|\,|B|\,|C|. \end{aligned}$$

The extension of (1.17) to any number of sets allows us to obtain other useful formulas for counting tuples of things. For example, for any finite set A and any natural number n we have the following product rule.

$$|A^n| = |A|^n. \qquad (1.18)$$

Counting Strings as Tuples

We can use product rules to count strings as well as tuples because a string can be represented as a tuple. In each of the following examples, the problem to be solved is expressed in terms of strings.

example

1.30 Counting All Strings

Suppose we need to count the number of strings of length 5 over the alphabet $A = \{a, b, c\}$. Any string of length 5 can be considered as a 5-tuple. For example, the string *abcbc* can be represented by the tuple (a, b, c, b, c). So the number of strings of length 5 over A equals the number of 5-tuples over A, which by product rule (1.18) is

$$|A^5| = |A|^5 = 3^5 = 243.$$

end example

example

1.31 Strings with Restrictions

Suppose we need to count the number of strings of length 6 over the alphabet $A = \{a, b, c, d\}$ that begin with either a or c and contain at least one occurrence of b.

Since strings can be represented by tuples, we'll count the number of 6-tuples over A that begin with either a or c and contain at least one occurrence of b. We'll break up the problem into two simpler problems. First, let U be the set of 6-tuples over A that begin with a or c. In other words, $U = \{a, c\} \times A^5$. Next, let S be the subset of U consisting of those 6-tuples that do not contain any occurrences of b. In other words, $S = \{a, c\} \times \{a, c, d\}^5$. Then the set $U - S$ is the desired set of 6-tuples over A that begin with either a or c and contain at least one occurrence of b. So we have

$$\begin{aligned} |U - S| &= |U| - |S| && \text{by (1.14)} \\ &= |\{a,c\} \times A^5| - |\{a,c\} \times \{a,c,d\}^5| \\ &= |\{a,c\}||A|^5 - |\{a,c\}||\{a,c,d\}|^5 && \text{(1.17) and (1.18)} \\ &= 2(4^5) - 2(3^5) \\ &= 1,562. \end{aligned}$$

end example

example

1.32 Strings with More Restrictions

We'll count the number of strings of length 6 over $A = \{a, b, c, d\}$ that start with a or c and contain at least one occurrence of either b or d.

As in the previous example, let U be the set of 6-tuples over A that start with a or c. Then $U = \{a, c\} \times A^5$ and $|U| = 2(4^5)$. Now let S be the subset of U whose 6-tuples do not contain any occurrences of b and do not contain any occurrences of d. So $S = \{a, c\}^6$ and $|S| = 2^6$. Then the set $U - S$ is the desired set of 6-tuples over A that begin with either a or c and contain at least one occurrence of either b or d. So by (1.14) we have

$$\begin{aligned} |U - S| &= |U| - |S| \\ &= 2(4^5) - 2^6 \\ &= 1,984. \end{aligned}$$

end example

example **1.33 Strings with More Restrictions**

Suppose we need to count the number of strings of length 6 over $A = \{a, b, c, d\}$ that start with a or c and contain at least one occurrence of b and at least one occurrence of d.

In this case we'll break up the problem into three simpler problems. First, let U be the set of 6-tuples that start with a or c. So $U = \{a, c\} \times A^5$ and $|U| = 2(4^5)$. Next, let S be the subset of U whose 6-tuples don't contain b. So $S = \{a, c\} \times \{a, c, d\}^5$ and $|S| = 2(3^5)$. Similarly, let T be the subset of U whose 6-tuples don't contain d. So $T = \{a, c\} \times \{a, b, c\}^5$ and $|T| = 2(3^5)$. Then the set $U - (S \cup T)$ is the desired set of 6-tuples that start with a or c and contain at least one occurrence of b and at least one occurrence of d. The cardinality of this set has the form

$$\begin{aligned} |U - (S \cup T)| &= |U| - |S \cup T| && \text{(by 1.14)} \\ &= |U| - (|S| + |T| - |S \cap T|). && \text{(by 1.11)} \end{aligned}$$

We'll be done if we can calculate the cardinality of $S \cap T$. Notice that

$$\begin{aligned} S \cap T &= \{a,c\} \times \{a,c,d\}^5 \cap \{a,c\} \times \{a,b,c\}^5 \\ &= \{a,c\} \times \{a,c\}^5 \\ &= \{a,c\}^6. \end{aligned}$$

So $|S \cap T| = 2^6$. Now we can complete the calculation of $|U - (S \cup T)|$.

$$\begin{aligned} |U - (S \cup T)| &= |U| - (|S| + |T| - |S \cap T|) \\ &= 2(4^5) - [2(3^5) + 2(3^5) - 2^6] \\ &= 1,140. \end{aligned}$$

end example

Exercises

Tuples

1. Write down all possible 3-tuples over the set $\{x, y\}$.

2. Let $A = \{a, b, c\}$ and $B = \{a, b\}$. Compute each of the following sets.

   a. $A \times B$. b. $B \times A$. c. A^0.
   d. A^1. e. A^2. f. $A^2 \cap (A \times B)$.

Lists

3. Write down all possible lists of length 2 or less over the set $A = \{a, b\}$.

4. Find the head and tail of each list.

   a. $\langle a \rangle$. b. $\langle a, b, c \rangle$. c. $\langle \langle a, b \rangle, c \rangle$.
   d. $\langle \langle a, b \rangle, \langle a, c \rangle \rangle$.

5. For positive integers m and n let D be the list of integers greater than 1 that divide both m and n, where D is ordered from smallest to largest. For example, if $m = 12$ and $n = 18$, then $D = \langle 2, 3, 6 \rangle$. We'll combine this information into a list $\langle m, n, D \rangle = \langle 12, 18, \langle 2, 3, 6 \rangle \rangle$. Construct $\langle m, n, D \rangle$ for each of the following cases.

   a. $m = 24$ and $n = 60$.
   b. $m = 36$ and $n = 36$.
   c. $m = 14$ and $n = 15$.

6. Write down all possible lists over $\{a, b\}$ that can be represented with five symbols, where the symbols that we count are a or b or $\langle$ or $\rangle$. For example, $\langle a, \langle \, \rangle \rangle$ uses five of the symbols. Can you do the same for lists over A that have six symbols? There are quite a few of them.

Strings and Languages

7. Write down all possible strings of length 2 over the set $A = \{a, b, c\}$.

8. Let $L = \{\Lambda, abb, b\}$ and $M = \{bba, ab, a\}$. Evaluate each of the following language expressions.

   a. LM. b. ML. c. L^0. d. L^1. e. L^2.

9. Use your wits to solve each of the following language equations for the unknown language.

   a. $\{\Lambda, a, ab\} L = \{b, ab, ba, aba, abb, abba\}$.
   b. $L\{a, b\} = \{a, baa, b, bab\}$.
   c. $\{a, aa, ab\} L = \{ab, aab, abb, aa, aaa, aba\}$.
   d. $L\{\Lambda, a\} = \{\Lambda, a, b, ab, ba, aba\}$.
   e. $\{a, b\} L = \{a, b, aba, bba\}$.
   f. $L\{b, \Lambda, ab\} = \{abb, ab, abab, bab, b, bb\}$.

10. Let L and M be two languages. For each of the following languages describe the general structure of a string x by writing it as a concatenation of strings that are in either L or M.

    a. LML. b. LM^*. c. $(L \cup M)^*$.
    d. $(L \cap M)^*$. e. L^*M^*. f. $(LM)^*$.

11. Try to describe each of the following languages in some way.

    a. $\{a, b\}^* \cap \{b, c\}^*$ b. $\{a, b\}^* - \{b\}^*$. c. $\{a, b, c\}^* - \{a\}^*$.

Relations

12. Represent each relation as a set by listing each individual tuple.

    a. $\{(d, 12) \mid d > 0$ and d divides $12\}$.
    b. $\{(d, n) \mid d, n \in \{2, 3, 4, 5, 6\}$ and d divides $n\}$.
    c. $\{(x, y, z) \mid x = y + z$, where $x, y, z \in \{1, 2, 3\}\}$.
    d. Let $(x, y) \in S$ if and only if $x \leq y$ and $x, y \in \{1, 2, 3\}$.
    e. Let $(x, y) \in U$ if and only if $x \in \{a, b\}$ and $y \in \{1, 2\}$.

13. Each of the following database queries refers to the Borders relational database given prior to Example 1.29. Express each answer by defining a set or relation in the same manner as the answer given in Example 1.29.

    a. What bodies of water border Michigan?
    b. What states border the Pacific Ocean?
    c. What states are landlocked?
    d. What relation represents the state-water pairs?
    e. What state-water pairs contain a state bordering Canada?

Counting Tuples

14. For each of the following cases, find the number of strings over the alphabet $\{a, b, c, d, e\}$ that satisfy the given conditions.

a. Length 4, begins with a or b, contains at least one c.
b. Length 5, begins with a, ends with b, contains at least one c or d.
c. Length 6, begins with d, ends with b or d, contains no c's.
d. Length 6, contains at least one a and at least one b.

15. Find a formula for the number of strings of length n over an alphabet A such that each string contains at least one occurrence of a letter from a subset B of A. Express the answer in terms of $|A|$ and $|B|$.

Proofs and Challenges

16. Prove each of the following statements about combining set operations with Cartesian product.

a. $(A \cup B) \times C = (A \times C) \cup (B \times C)$.
b. $(A - B) \times C = (A \times C) - (B \times C)$.
c. Find and prove a similar equality using the intersection operation.

17. Let L, M, and N be languages. Prove each of the following properties of the product operation on languages.

a. $L\{\Lambda\} = \{\Lambda\}L = L$.
b. $L\varnothing = \varnothing L = \varnothing$.
c. $L(M \cup N) = LM \cup LN$ and $(M \cup N)L = ML \cup NL$.
d. $L(M \cap N) \subset LM \cap LN$ and $(M \cap N)L \subset ML \cap NL$.

18. Let L and M be languages. Prove each of the following statements about the closure of languages (1.16).

a. $\{\Lambda\}^* = \varnothing^* = \{\Lambda\}$.
b. $\Lambda \in L$ if and only if $L^+ = L^*$.
c. $L^* = L^*L^* = (L^*)^*$.
d. $(L^*M^*)^* = (L^* \cup M^*)^* = (L \cup M)^*$.

19. (*Tuples Are Special Sets*). We can define the concept of tuples in terms of sets. For example, we'll define

$$(\,) = \varnothing,\ (x) = \{x\}, \quad \text{and} \quad (x, y) = \{\{x\}, \{x, y\}\}.$$

Use this definition to verify each of the following statements.

a. Show that $(3, 7) \neq (7, 3)$.
b. Show that $(x, y) = (u, v)$ if and only if $x = u$ and $y = v$.
c. Find an example to show that the definition $(x, y) = \{x, \{y\}\}$ will not distinguish between distinct 2-tuples.

20. (*Tuples Are Special Sets*). Continuing with Exercise 19, we can define a 3-tuple in terms of sets by letting S be the set representing the ordered pair (x, y) from Exercise 19. Then define

$$(x, y, z) = \{\{S\}, \{S, z\}\}.$$

   a. Write down the complete set to represent (x, y, z).
   b. Show that $(a, b, c) = (d, e, f)$ if and only if $a = d$, $b = e$, and $c = f$.
   c. Find an example to show that the definition

$$(x, y, z) = \{\{x\}, \{x, y\}, \{x, y, z\}\}$$

   will not distinguish between distinct 3-tuples.

   Note: We could continue in this manner and define n-tuples as sets for any natural number n. Although defining a tuple as a set is not at all intuitive, it does illustrate how sets can be used as a foundation from which to build objects and ideas. It also shows why good notation is so important for communicating ideas.

21. Use Example 1.23 as a guide to find the address polynomial for an arbitrary element in each of the following cases. Assume that all indexes start with 1, that the beginning address is B, and each component needs M memory cells.
   a. A matrix of size 3 by 4 stored in column-major form as a 4-tuple of columns, each of which is a 3-tuple.
   b. A matrix of size m by n stored in row-major form.
   c. A three-dimensional array of size l by m by n stored as an l-tuple where each component is an m by n matrix stored in row-major form.

1.4 Graphs and Trees

When we think about graphs we might think about pictures of some kind that are used to represent information. The graphs that we'll discuss can be thought about in the same way. But we need to describe them in a little more detail if they are to be much use to us. We'll also see that trees are special kinds of graphs.

1.4.1 Definition of a Graph

Informally, a *graph* is a set of objects in which some of the objects are connected to each other in some way. The objects are called *vertices* or *nodes*, and the

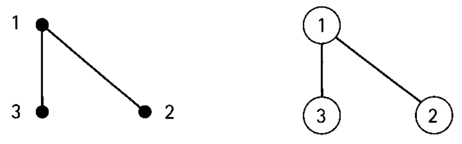


Figure 1.11 Graphs.

connections are called *edges*. For example, the United States can be represented by a graph where the vertices are states and the edges are the common borders between adjacent states. In this case, Hawaii and Alaska would be vertices without any edges connected to them. We say that two vertices are *adjacent* if there is an edge connecting them.

Picturing a Graph

We can picture a graph in several ways. For example, Figure 1.11 shows two ways to represent the graph with vertices 1, 2, and 3 and edges connecting 1 to 2 and 1 to 3.

example **1.34 States and Provinces**

Figure 1.12 represents a graph of those states in the United States and those provinces in Canada that touch the Pacific Ocean or that touch states and provinces that touch the Pacific Ocean.

end example

Coloring a Graph

An interesting problem dealing with maps is to try to color a map with the fewest number of colors subject to the restriction that any two adjacent areas must have distinct colors. From a graph point of view, this means that any two distinct adjacent vertices must have different colors. Before reading any further, try to color the graph in Figure 1.12 with the fewest colors. It's usually easier to represent the colors by numbers like 1, 2, ...

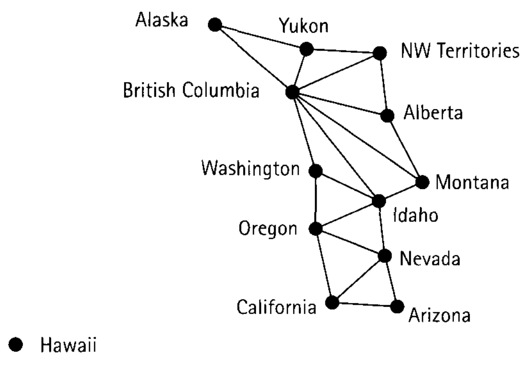


Figure 1.12 Graph of states and provinces.

A graph is *n-colorable* if there is an assignment of n colors to its vertices such that any two distinct adjacent vertices have distinct colors. The *chromatic number* of a graph is the smallest n for which it is n-colorable. For example, the chromatic number of the graph in Figure 1.12 is 3. A graph whose edges are the connections between all pairs of distinct vertices is called a *complete graph.* It's easy to see that the chromatic number of a complete graph with n vertices is n.

A graph is *planar* if it can be drawn on a plane such that no edges intersect. For example, the graph in Figure 1.12 is planar. A complete graph with four vertices is planar, but a complete graph with five vertices is not planar. See whether you can convince yourself of these facts. A fundamental result on graph coloring—which remained an unproven conjecture for over 100 years—states that *every planar graph is 4-colorable.* The result was proven in 1976 by Kenneth Appel and Wolfgang Haken. They used a computer to test over 1900 special cases. For example, see Appel and Haken [1976, 1977].

More Terminology

A *directed graph* (*digraph* for short) is a graph where each edge points in one direction. For example, the vertices could be cities and the edges could be the one-way air routes between them. For digraphs we use arrows to denote the edges. For example, Figure 1.13 shows two ways to represent the digraph with three vertices a, b, and c and edges from a to b, c to a, and c to b.

The *degree* of a vertex is the number of edges that it touches. However, we add two to the degree of a vertex if it has a *loop*, which is an edge that starts and ends at the same vertex. For directed graphs the *indegree* of a vertex is the number of edges pointing at the vertex, whereas the *outdegree* of a vertex is the number of edges pointing away from the vertex. In a digraph a vertex is called a *source* if its indegree is zero and a *sink* if its outdegree is zero. For example, in the digraph of Figure 1.13, c is a source and b is a sink.

If a graph has more than one edge between some pair of vertices, the graph is called a *multigraph*, or a *directed multigraph* in case the edges point in the same direction. For example, there are usually two or more road routes between most cities. So a graph representing road routes between a set of cites is most likely a multigraph.

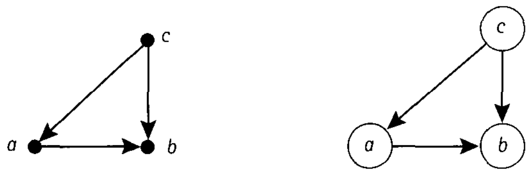


Figure 1.13 Directed graphs.

Representaions of Graphs

From a computational point of view, we need to represent graphs as data. This is easy to do because we can define a graph in terms of tuples, sets, and bags. For example, we can define a graph G as an ordered pair (V, E), where V is a set of vertices and E is a set or bag of edges. If G is a digraph, then the edges in E can be represented by ordered pairs, where (a, b) represents the edge with an arrow from a to b. In this case the set E of edges is a subset of $V \times V$. In other words, E is a binary relation on V. For example, the digraph in Figure 1.13 has vertex set $\{a, b, c\}$ and edge set

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

If G is a directed multigraph, then we can represent the edges as a bag (or multiset) of ordered pairs. For example, the bag $[(a, b), (a, b), (b, a)]$ represents three edges: two from a to b and one from b to a.

If a graph is not directed, we have more ways to represent the edges. We could still represent an edge as an ordered pair (a, b) and agree that it represents an undirected line between a and b. But we can also represent an edge between vertices a and b by a set $\{a, b\}$. For example, the graph in Figure 1.11 has vertex set $\{1, 2, 3\}$ and edge set $\{\{1, 2\}, \{1, 3\}\}$.

Weighted Graphs

We often encounter graphs that have information attached to each edge. For example, a good road map places distances along the roads between major intersections. A graph is called *weighted* if each edge is assigned a number, called a *weight*. We can represent an edge (a, b) that has weight w by the 3-tuple

$$(a, b, w).$$

In some cases we might want to represent an unweighted graph as a weighted graph. For example, if we have a multigraph in which we wish to distinguish between multiple edges that occur between two vertices, then we can assign a different weight to each edge, thereby creating a weighted multigraph.

Graphs and Binary Relations

We can observe from our discussion of graphs that any binary relation R on a set A can be thought of as a digraph $G = (A, R)$ with vertices A and edges R. For example, let $A = \{1, 2, 3\}$ and

$$R = \{(1, 2), (1, 3), (2, 3), (3, 3)\}.$$

Figure 1.14 shows the digraph corresponding to this binary relation. Representing a binary relation as a graph is often quite useful in trying to establish properties of the relation.

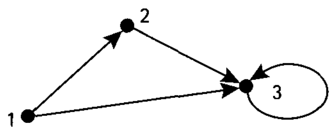


Figure 1.14 Digraph of binary relation.

Subgraphs

Sometimes we need to discuss graphs that are part of other graphs. A graph (V', E') is a *subgraph* of a graph (V, E) if $V' \subset V$ and $E' \subset E$. For example, the four graphs in Figure 1.15 are subgraphs of the graph in Figure 1.14.

1.4.2 Paths in Graphs

Problems that use graphs often involve moving from one vertex to another along a sequence of edges, where each edge shares a vertex with the next edge in the sequence. In formal terms, a *path* from x_0 to x_n is a sequence of edges that we denote by a sequence of vertices $x_0, x_1, \ldots, x_n$ such that there is an edge from x_{i-1} to x_i for $1 \leq i \leq n$. A path allows the possibility that some edge or some vertex occurs more than once. A *cycle* is a path whose beginning and ending vertices are equal and in which no edge occurs more than once. A graph with no cycles is called *acyclic*. The *length* of the path $x_0, \ldots, x_n$ is the number n of edges.

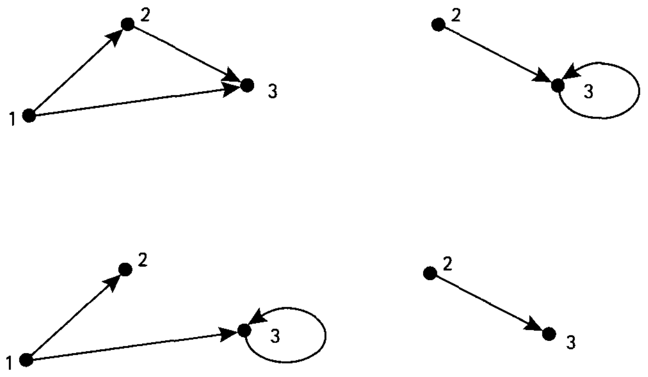


Figure 1.15 Subgraphs.

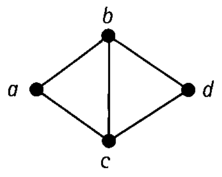


Figure 1.16 Sample graph.

example

1.35 Paths in a Graph

We'll examine a few paths in the graph pictured in Figure 1.16.

1. The path b, c, d, b, a visits b twice. The length of the path is 4.
2. The path a, b, c, b, d visits b twice and uses the edge between b and c twice. The length of the path is 4.
3. The path a, b, c, a is a cycle of length 3.
4. The path a, b, a is not a cycle because the edge from a to b occurs twice. The path has length 2.

end example

Connected Graphs

Let's emphasize here that the definitions for path and cycle apply to both graphs and directed graphs. But now we come to an idea that needs a separate definition for each type of graph. A graph is *connected* if there is a path between every pair of vertices. A directed graph is *connected* if, when direction is ignored, the resulting undirected graph is connected.

Two Graph Problems

Let's look at two graph problems that you may have seen. For the first problem you are to trace the first diagram in Figure 1.17 without taking your pencil off the paper and without retracing any line.

After some fiddling, it's easy to see that it can be done by starting at one of the bottom two corners and finishing at the other bottom corner. The second diagram in Figure 1.17 emphasizes the graphical nature of the problem. From this point of view we can say that there is a path that travels each edge exactly once.

The second famous problem is named after the seven bridges of Königsberg that, in the early 1800s, connected two islands in the river Pregel to the rest of the town. The problem is to tour the town by walking across each of the seven

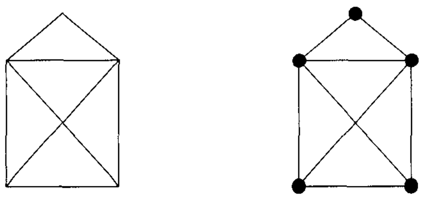


Figure 1.17 Tracing a graph.

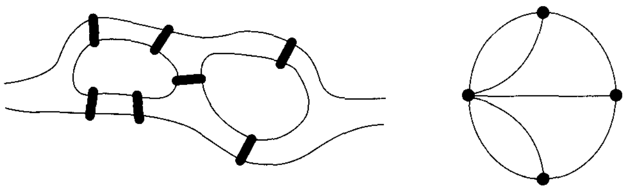


Figure 1.18 Seven bridges of Königsberg.

bridges exactly once. In Figure 1.18 we've pictured the two islands and seven bridges of Königsberg together with a multigraph representing the situation. The vertices of the multigraph represent the four land areas and the edges represent the seven bridges.

The mathematician Leonhard Euler (1707–1783) proved that there aren't any such paths by finding a general condition for such paths to exist. In his honor, any path that contains each edge of a graph exactly once is called an *Euler path.* For example, the graph for the tracing problem has an Euler path, but the seven bridges problem does not. We can go one step further and define an *Euler circuit* to be any path that begins and ends at the same vertex and contains each edge of a graph exactly once. There are no Euler circuits in the graphs of Figure 1.17 and Figure 1.18. We'll discuss conditions for the existence of Euler paths and Euler circuits in the exercises.

1.4.3 Graph Traversals

A *graph traversal* starts at some vertex v and visits all vertices x that can be reached from v by traveling along some path from v to x. If a vertex has already been visited, it is not visited again. Two popular traversal algorithms are called *breadth-first* and *depth-first.*

Breadth-First Traversal

To describe *breadth-first traversal,* we'll let visit(v, k) denote the procedure that visits every vertex x not yet visited for which there is a length k path from v to x. If the graph has n vertices, a breadth-first traversal starting at v can be

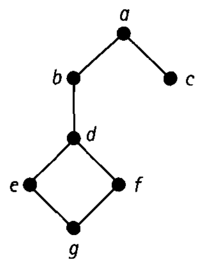


Figure 1.19 Sample graph.

described as follows:

$$\textbf{for } k := 0 \textbf{ to } n - 1 \textbf{ do } \text{visit}(v,\ k) \textbf{ od}.$$

Since we haven't specified how visit(v, k) does it's job, there are usually several different traversals from any given starting vertex.

example

1.36 Breadth-First Traversals

We'll do some breadth-first traversals of the graph in Figure 1.19. If we start at vertex a, then there are four possible breadth-first traversals, which we've represented by the following strings:

$a\ b\ c\ d\ e\ f\ g$ $\qquad$ $a\ b\ c\ d\ f\ e\ g$ $\qquad$ $a\ c\ b\ d\ e\ f\ g$ $\qquad$ $a\ c\ b\ d\ f\ e\ g.$

Of course, we can start a breadth-first traversal at any vertex. For example, one of several breadth-first traversals that start with d is represented by the string

$$d\ b\ e\ f\ a\ g\ c.$$

end example

Depth-First Traversal

We can describe *depth-first traversal* with a recursive procedure—one that calls itself. Let DF(v) denote the depth-first procedure that traverses the graph starting at vertex v. Then DF(v) has the following definition.

```
DF($v$):  if $v$ has not been visited then
              visit $v$;
              for each edge from $v$ to $x$ do DF($x$) od
          fi
```

Since we haven't specified how each edge from v to x is picked in the **for** loop, there are usually several different traversals from any given starting vertex.

example **1.37 Depth-First Traversals**

We'll do some depth-first traversals of the graph represented in Figure 1.19. For example, starting from vertex a in the graph, there are four possible depth-first traversals, which are represented by the following strings:

$$a\ b\ d\ e\ g\ f\ c \qquad a\ b\ d\ f\ g\ e\ c \qquad a\ c\ b\ d\ e\ g\ f \qquad a\ c\ b\ d\ f\ g\ e.$$

end example

1.4.4 Trees

From an informal point of view, a *tree* is a structure that looks like a real tree. For example, a family tree and an organizational chart for a business are both trees. From a formal point of view we can say that a *tree* is a graph that is connected and has no cycles. Such a graph can be drawn to look like a real tree. The vertices and edges of a tree are called *nodes* and *branches*, respectively.

In computer science, and some other areas, trees are usually pictured as upside down versions of real trees, as in Figure 1.20. For trees represented this way, the node at the top is called the *root*. The nodes that hang immediately below a given node are its *children*, and the node immediately above a given node is its *parent*. If a node is childless, then it is a *leaf*. The *height* or *depth* of a tree is the length of the longest path from the root to the leaves. The path from a node to the root contains all the *ancestors* of the node. Any path from a node to a leaf contains *descendants* of the node.

A tree with a designated root is often called a *rooted tree*. Otherwise, it is called a *free tree* or an *unrooted tree*. We'll use the term *tree* and let the context indicate the type of tree.

example **1.38 Parts of a Tree**

We'll make some observations about the tree in Figure 1.20. The root is A. The children of A are B, C, and D. The parent of F is B. The leaves of the tree are are E, F, J, H, and I. The height or depth of the tree is 3.

end example

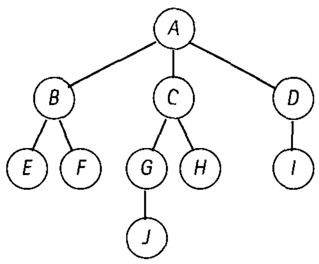


Figure 1.20 Sample tree.

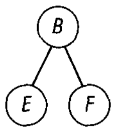


Figure 1.21 A subtree.

Subtrees

If x is a node in a tree T, then x together with all its descendants forms a tree S with x as its root. S is called a *subtree* of T. If y is the parent of x, then S is sometimes called a subtree of y.

example 1.39 A Subtree

The tree pictured in Figure 1.21 is a subtree of the tree in Figure 1.20. Since A is the parent of B, we can also say that this tree is a subtree of node A.

end example

Ordered and Unordered Trees

If we don't care about the ordering of the children of a tree, then the tree is called an *unordered tree.* A tree is *ordered* if there is a unique ordering of the children of each node. For example, any algebraic expression can be represented as ordered tree.

example 1.40 Representing Algebraic Expressions

The expression $x - y$ can be represented by a tree whose root is the minus sign and with two subtrees, one for x on the left and one for y on the right. Ordering is important here because the subtraction operation is not commutative. For example, Figure 1.22 shows the expression $3 - (4 + 8)$ and Figure 1.23 shows the expression $(4 + 8) - 3$.

end example

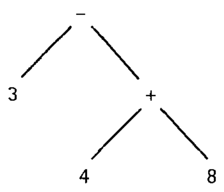


Figure 1.22 Tree for $3 - (4 + 8)$.

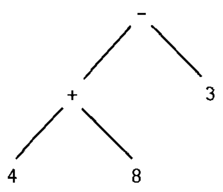


Figure 1.23 Tree for $(4 + 8) - 3$.

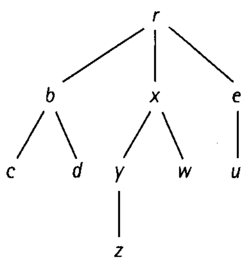


Figure 1.24 Sample tree.

Representations of Trees

How can we represent a tree as a data object? The key to any representation is that we should be able to recover the tree from its representation. One method is to let the tree be a list whose first element is the root and whose next elements are lists that represent the subtrees of the root.

For example, the tree with a single node r is represented by $\langle r \rangle$, and the list representation of the tree for the algebraic expression $a - b$ is

$$\langle -, \langle a \rangle, \langle b \rangle \rangle.$$

For another example, the list representation of the tree for the arithmetic expression $3 - (4 + 8)$ is

$$\langle -, \langle 3 \rangle, \langle +, \langle 4 \rangle, \langle 8 \rangle \rangle \rangle.$$

For a more complicated example, let's consider the tree represented by the following list.

$$T = \langle r, \langle b, \langle c \rangle, \langle d \rangle \rangle, \langle x, \langle y, \langle z \rangle \rangle, \langle w \rangle \rangle, \langle e, \langle u \rangle \rangle \rangle.$$

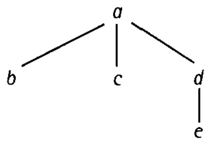
Notice that T has root r, which has the following three subtrees:

$$\langle b, \langle c \rangle, \langle d \rangle \rangle$$
$$\langle x, \langle y, \langle z \rangle \rangle, \langle w \rangle \rangle$$
$$\langle e, \langle u \rangle \rangle.$$

Similarly, the subtree $\langle b, \langle c \rangle, \langle d \rangle \rangle$ has root b, which has two children c and d. We can continue in this way to recover the picture of T in Figure 1.24.

example 1.41 Computer Repesentation of Trees

Let's represent a tree as a list and then see what it looks like in computer memory. For example, let T be the following tree.



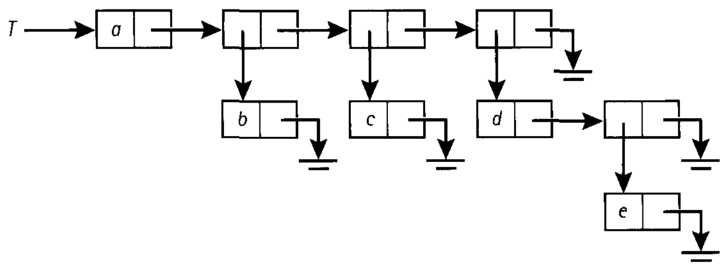


Figure 1.25 Computer representation of a tree.

We'll represent the tree as the list $T = \langle a, \langle b\rangle, \langle c\rangle, \langle d, \langle e\rangle\rangle\rangle$. Figure 1.25 shows the representation of T in computer memory, where we have used the same notation as Example 1.24.

end example

Binary Trees

A *binary tree* is an ordered tree that may be empty or else has the property that each node has two subtrees, called the *left* and *right* subtrees of the node, which are binary trees. We can represent the empty binary tree by the empty list $\langle\ \rangle$. Since each node has two subtrees, we represent nonempty binary trees as 3-element lists of the form

$$\langle L, x, R\rangle,$$

where x is the root, L is the left subtree, and R is the right subtree. For example, the tree with one node x is represented by the list $\langle\langle\ \rangle, x, \langle\ \rangle\rangle$.

When we draw a picture of a binary tree, it is common practice to omit the empty subtrees. For example, the binary tree represented by the list

$$\langle\langle\langle\ \rangle, a, \langle\ \rangle\rangle, b, \langle\ \rangle\rangle$$

is usually, but not always, pictured as the simpler tree in Figure 1.26.

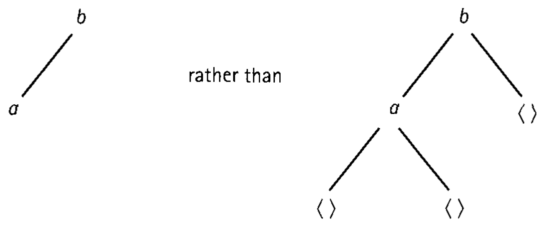


Figure 1.26 Simplified binary tree.

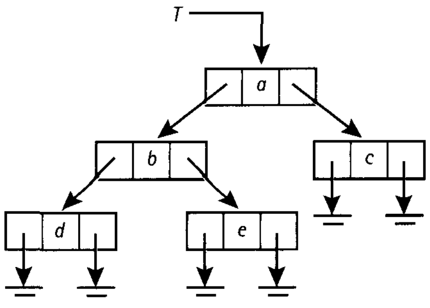


Figure 1.27 Computer representation of a binary tree.

example **1.42 Computer Repesentation of Binary Trees**

Let's see what the representation of a binary tree as a list looks like in computer memory. For example, let T be the following tree.

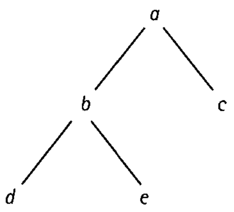


Figure 1.27 shows the representation of T in computer memory, where each block of memory contains a node and pointers to the left and right subtrees.

end example

Binary Search Trees

Binary trees can be used to represent sets whose elements have some ordering. Such a tree is called a *binary search tree* and has the property that for each node of the tree, each element in its left subtree precedes the node element and each element in its right subtree succeeds the node element.

example **1.43 A Binary Search Tree**

The binary search tree in Figure 1.28 holds three-letter abbreviations for six of the months, where we are using the dictionary ordering of the words. So the correct order is FEB, JAN, JUL, NOV, OCT, SEP.

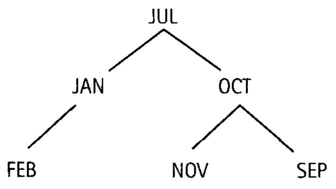


Figure 1.28 Binary search tree.

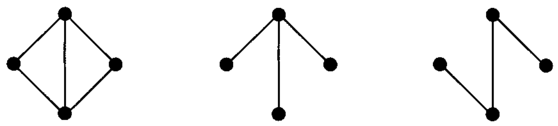


Figure 1.29 Two spanning trees.

This binary search tree has depth 2. There are many other binary search trees to hold these six months. Find another one that has depth 2. Then find one that has depth 3.

end example

1.4.5 Spanning Trees

A *spanning tree* for a connected graph is a subgraph that is a tree and contains all the vertices of the graph. For example, Figure 1.29 shows a graph followed by two of its spanning trees. This example shows that a graph can have many spanning trees. A *minimal spanning tree* for a connected weighted graph is a spanning tree such that the sum of the edge weights is minimum among all spanning trees.

Prim's Algorithm

A famous algorithm, due to Prim [1957], constructs a minimal spanning tree for any undirected connected weighted graph. Starting with any vertex, the algorithm searches for an edge of minimum weight connected to the vertex. It adds the edge to the tree and then continues by trying to find new edges of minimum weight such that one vertex is in the tree and the other vertex is not. Here's an informal description of the algorithm.

Prim's Algorithm **(1.19)**

Construct a minimal spanning tree for an undirected connected weighted graph. The variables: V is the vertex set of the graph; W is the vertex set and S is the edge set of the spanning tree.

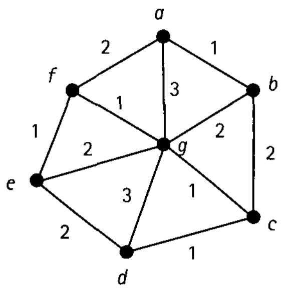
Continued ➧

```
1. Initialize $S := \varnothing$.
2. Pick any vertex $v \in V$ and set $W := \{v\}$.
3. while $W \neq V$ do
       Find a minimum weight edge $\{x, y\}$, where $x \in W$ and
         $y \in V - W$;
       $S := S \cup \{\{x, y\}\}$;
       $W := W \cup \{y\}$
   od
```

Of course, Prim's algorithm can also be used to find a spanning tree for an unweighted graph. Just assign a weight of 1 to each edge of the graph. Or modify the first statement in the **while** loop to read "Find an edge $\{x, y\}$ such that $x \in W$ and $y \in V - W$."

example 1.44 A Minimal Spanning Tree

We'll construct a minimal spanning tree for the following weighted graph.



To see how the algorithm works we'll do a trace of each step showing the values of the variables S and W. The algorithm gives us several choices since it is not implemented as a computer program. So we'll start with the letter a since it's the first letter of the alphabet.

S	W
$\{\}$	$\{a\}$
$\{\{a, b\}\}$	$\{a, b\}$
$\{\{a, b\}, \{b, c\}\}$	$\{a, b, c\}$
$\{\{a, b\}, \{b, c\}, \{c, d\}\}$	$\{a, b, c, d\}$
$\{\{a, b\}, \{b, c\}, \{c, d\}, \{c, g\}\}$	$\{a, b, c, d, g\}$
$\{\{a, b\}, \{b, c\}, \{c, d\}, \{c, g\}, \{g, f\}\}$	$\{a, b, c, d, g, f\}$
$\{\{a, b\}, \{b, c\}, \{c, d\}, \{c, g\}, \{g, f\}, \{f, e\}\}$	$\{a, b, c, d, g, f, e\}$

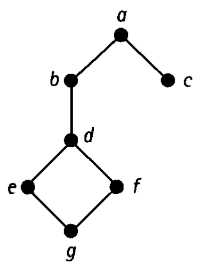
The algorithm stops because $W = V$. So S is a spanning tree.

end example

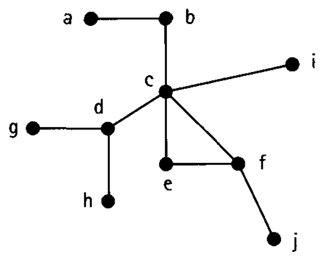
Exercises

Graphs

1. Draw a picture of a graph that represents those states of the United States and those provinces of Canada that touch the Atlantic Ocean or touch states or provinces that do.
2. Find planar graphs with the smallest possible number of vertices that have chromatic numbers of 1, 2, 3, and 4.
3. What is the chromatic number of the graph representing the map of the United States? Explain your answer.
4. Draw a picture of the directed graph that corresponds to each of the following binary relations.
   a. $\{(a, a), (b, b), (c, c)\}$.
   b. $\{(a, b), (b, b), (b, c), (c, a)\}$.
   c. The relation $\leq$ on the set $\{1, 2, 3\}$.
5. Given the following graph:

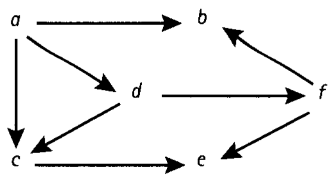


   a. Write down all breadth-first traversals that start at vertex *f*.
   b. Write down all depth-first traversals that start at vertex *f*.
6. Given the following graph.



a. Write down one breadth-first traversal that starts at vertex *f*.
b. Write down one depth-first traversal that starts at vertex *f*.

7. Given the following graph:



a. Write down one breadth-first traversal that starts at vertex *a*.
b. Write down one depth-first traversal that starts at vertex *a*.

Trees

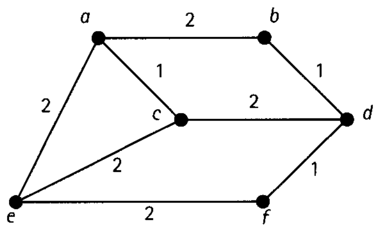
8. Given the algebraic expression $a \times (b + c) - (d \ / \ e)$. Draw a picture of the tree representation of this expression. Then convert the tree into a list representation of the expression.

9. Draw a picture of the ordered tree that is represented by the list

$$\langle a, \langle b, \langle c \rangle, \langle d, \langle e \rangle \rangle \rangle, \langle r, \langle s \rangle, \langle t \rangle \rangle, \langle x \rangle \rangle.$$

10. Draw a picture of a binary search tree containing the three-letter abbreviations for the 12 months of the year in dictionary order. Make sure that your tree has the least possible depth.

11. Find two distinct minimal spanning trees for the following weighted graph.



12. For the weighted graph in Example 1.44, find two distinct minimal spanning trees that are different from the spanning tree given.

Path Problems

13. Try to find a necessary and sufficient condition for an undirected graph to have an Euler path. *Hint:* Look at the degrees of the vertices.

14. Try to find a necessary and sufficient condition for an undirected graph to have an Euler circuit. *Hint:* Look at the degrees of the vertices.

1.5 Chapter Summary

We normally prove things informally, and we use a variety of proof techniques: proof by exhaustive checking, conditional proof, indirect proofs (i.e., proving the contrapositive and proof by contradiction), and iff proofs.

Sets are characterized by lack of order and no repeated occurrences of elements. There are easy techniques for comparing sets by subset and by equality. Sets can be combined by the operations of union, intersection, difference, and complement. Venn diagrams are useful for representing these operations. Two useful rules for counting sets are the union rule—also called the inclusion-exclusion principle—and the difference rule.

Bags—also called multisets—are characterized by lack of order, and they may contain repeated occurrences of elements.

Tuples are characterized by order, and they may contain repeated occurrences of elements. Many useful structures are related to tuples. Cartesian products of sets are collections of tuples. Lists are similar to tuples except that lists can be accessed only by head and tail. Strings are like lists except that elements from an alphabet are placed next to each other in juxtaposition. Languages are sets of strings and they can be combined by concatenating strings as well as by set operations. Relations are sets of tuples that are related in some way. A useful rule for counting tuples is the product rule.

Graphs are characterized by a set of vertices and a set of edges, where the edges may be undirected or directed. Graphs can be colored, and they can be traversed. Trees are special graphs that look like real trees. Prim's algorithm constructs a minimal spanning tree for an undirected, connected, weighted graph.

chapter 2

Facts about Functions

All my discoveries were simply improvements in notation.
—Gottfried Wilhelm von Leibniz (1646–1716)[1]

Functions can often make life simpler. In this chapter we'll start with the basic notions and notations for functions. Then we'll introduce some functions that are especially important in computer science. Since programs can be functions and functions can be programs, we'll spend some time discussing techniques for constructing new functions from simpler ones. We'll also discuss other properties of functions that are useful in problem solving.

chapter guide

Section 2.1 introduces the basic ideas of functions—what they are, and how to represent them. We'll give many examples, including several functions that are especially useful to computer scientists.

Section 2.2 introduces the important idea of composition as a way to combine functions to construct new functions. We'll see that the map function is a useful tool for constructing functions that calculate lists.

Section 2.3 introduces three important properties of functions—injective, surjective, and bijective. We'll see how these properties are used when we discuss the pigeonhole principle, cryptology, and hash functions.

Section 2.4 gives a brief introduction to techniques for comparing infinite sets. We'll discuss the ideas of countable and uncountable sets. We'll introduce the diagonalization technique, and we'll discuss whether we can compute everything.

[1]Leibniz introduced the word "function" around 1692. He is responsible for such diverse ideas as binary arithmetic, symbolic logic, combinatorics, and calculus. Around 1694 he built a calculating machine that could add and multiply.

2.1 Definitions and Examples

In this section we'll give the definition of a function along with various ways to describe functions. We'll also spend some time with functions that are very useful in computer science.

2.1.1 Definition of a Function

Suppose A and B are sets and for each element in A we associate exactly one element in B. Such an association is called a *function* from A to B. The main idea is that each element of A is associated with *exactly one* element of B. In other words, if $x \in A$ is associated with $y \in B$, then x is not associated with any other element of B.

Functions are normally denoted by letters like f, g, and h or other descriptive names or symbols. If f is a function from A to B and f associates the element $x \in A$ with the element $y \in B$, then we write $f(x) = y$ or $y = f(x)$. The expression $f(x)$ is read, "f of x," or "f at x," or "f applied to x." When $f(x) = y$, we often say, "f maps x to y." Some other words for "function" are *mapping*, *transformation*, and *operator*.

Describing Functions

Functions can be described in many ways. Sometimes a formula will do the job. For example, the function f from $\mathbb{N}$ to $\mathbb{N}$ that maps every natural number x to its square can be described by the following formula:

$$f(x) = x^2.$$

Other times, we'll have to write down all possible associations. For example, the following associations define a function g from $A = \{a, b, c\}$ to $B = \{1, 2, 3\}$:

$$g(a) = 1, g(b) = 1, \text{ and } g(c) = 2.$$

We can also describe a function by a drawing a figure. For example, Figure 2.1 shows three ways to represent the function g. The top figure uses Venn diagrams together with a digraph. The lower-left figure is a digraph. The lower-right figure is the familiar Cartesian graph, in which each ordered pair $(x, g(x))$ is plotted as a point.

Figure 2.2 shows two associations that are not functions. Be sure to explain why these associations do not represent functions from A to B.

Terminology

To communicate with each other about functions, we need to introduce some more terminology. If f is a function from A to B, we denote this by writing

$$f : A \to B.$$

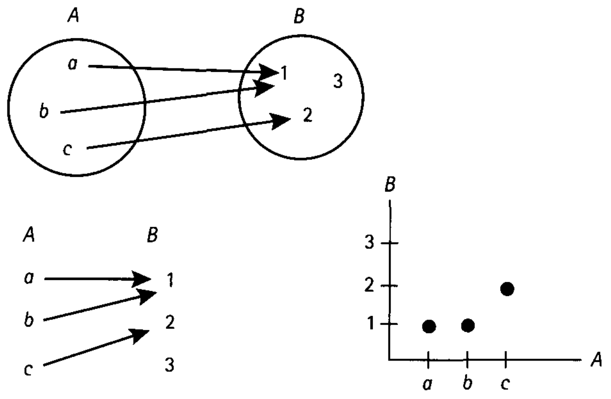


Figure 2.1 Three ways to describe the same function.

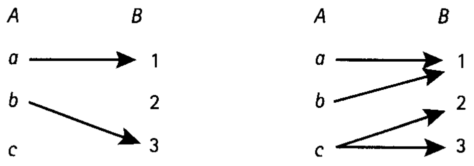


Figure 2.2 Two associations that are *not* functions.

The set A is the *domain* of f and the set B is the *codomain* of f. We also say that f has *type* $A \rightarrow B$. The expression $A \rightarrow B$ denotes the set of all functions from A to B.

If $f(x) = y$, then x is called an *argument* of f, and y is called a *value* of f. If the domain of f is the Cartesian product $A_1 \times \cdots \times A_n$, we say f has *arity* n or f has n *arguments*. In this case, if $(x_1, \ldots, x_n) \in A_1 \times \cdots \times A_n$, then

$$f(x_1, \ldots, x_n)$$

denotes the value of f at $(x_1, \ldots, x_n)$. A function f with two arguments is called a *binary* function and we have the option of writing $f(x, y)$ in the popular *infix* form $x\ f\ y$. For example, $4 + 5$ is usually preferable to $+(4, 5)$.

Ranges, Images, and Pre-Images

At times it is necessary to discuss certain subsets of the domain and codomain of a function $f : A \rightarrow B$. The *range* of f, denoted by range(f), is the set of elements in the codomain B that are associated with some element of A. In other words, we have

$$\text{range}(f) = \{f(a) \mid a \in A\} .$$

For any subset $S \subset A$, the *image* of S under f, denoted by $f(S)$, is the set of elements in B that are associated with some element of S. In other words, we have

$$f(S) = \{f(x) \mid x \in S\}.$$

Notice that we always have the special case $f(A) = \text{range}(f)$. Notice also that images allow us to think of f not only as a function from A to B, but also as a function from power(A) to power(B).

For any subset $T \subset B$ the *pre-image* of T under f, denoted by $f^{-1}(T)$, is the set of elements in A that associate with elements of T. In other words, we have

$$f^{-1}(T) = \{a \in A \mid f(a) \in T\}.$$

Notice that we always have the special case $f^{-1}(B) = A$. Notice also that pre-images allow us to think of f^{-1} as a function from power(B) to power(A).

example 2.1 Sample Notations

Consider the function $f : \{a, b, c\} \to \{1, 2, 3\}$ defined by $f(a) = 1$, $f(b) = 1$, and $f(c) = 2$. We can make the following observations.

f has type $\{a, b, c\} \to \{1, 2, 3\}$.
The domain of f is $\{a, b, c\}$.
The codomain of f is $\{1, 2, 3\}$.
The range of f is $\{1, 2\}$.

Some sample images are

$f(\{a\}) = \{1\}$,
$f(\{a, b\}) = \{1\}$,
$f(A) = f(\{a, b, c\}) = \{1, 2\} = \text{range}(f)$.

Some sample pre-images are

$f^{-1}(\{1, 3\}) = \{a, b\}$,
$f^{-1}(\{3\}) = \varnothing$,
$f^{-1}(B) = f^{-1}(\{1, 2, 3\}) = \{a, b, c\} = A$.

end example

example 2.2 Functions and Not Functions

Let P be the set of all people, alive or dead. We'll make some associations and discuss whether each is function of type $P \to P$.

1. $f(x)$ is a parent of x.

   In this case f is *not* a function of type $P \to P$ because people have two parents. For example, if q has mother m and father p, then $f(q) = m$ and $f(q) = p$, which is contrary to the requirement that each domain element be associated with exactly one codomain element.

2. $f(x)$ is the mother of x.

   In this case f *is* a function of type $P \to P$ because each person has exactly one mother. In other words, each $x \in P$ maps to exactly one person, the mother of x. If m is a mother, what is the pre-image of the set $\{m\}$ under f?

3. $f(x)$ is the oldest child of x.

   In this case f is *not* a function of type $P \to P$ because some person has no children. Therefore, $f(x)$ is not defined for some $x \in P$.

4. $f(x)$ is the set of all children of x.

   In this case f is *not* a function of type $P \to P$ because each person is associated with a set of people rather than a person. However, f *is* a function of type $P \to \text{power}(P)$. Can you see why?

end example

example

2.3 Tuples Are Functions

Any ordered sequence of objects can be thought of as a function. For example, the tuple (22, 14, 55, 1, 700, 67) can be thought of as a listing of the values of the function

$$f : \{0, 1, 2, 3, 4, 5\} \to \mathbb{N}$$

where f is defined by the equality

$$(f(0), f(1), f(2), f(3), f(4), f(5)) = (22, 14, 55, 1, 700, 67).$$

Similarly, any infinite sequence of objects can also be thought of as a function. For example, suppose that $(b_0, b_1, \ldots, b_n, \ldots)$ is an infinite sequence of objects from a set S. Then the sequence can be thought of as a listing of values in the range of the function $f : \mathbb{N} \to S$ defined by $f(n) = b_n$.

end example

example **2.4 Functions and Binary Relations**

Any function can be defined as a special kind of binary relation. A function $f : A \to B$ is a binary relation from A to B such that no two ordered pairs have the same first element. We can also describe this uniqueness condition as: If $(a, b), (a, c) \in f$, then $b = c$. The notation $f(a) = b$ is normally preferred over the relational notations $f(a, b)$ and $(a, b) \in f$.

end example

Equality of Functions

Two functions are equal if they have the same type and the same values for each domain element. In other words, if f and g are functions of type $A \to B$, then f and g are said to be *equal* if $f(x) = g(x)$ for all $x \in A$. If f and g are equal, we write

$$f = g.$$

For example, suppose f and g are functions of type $\mathbb{N} \to \mathbb{N}$ defined by the formulas $f(x) = x + x$ and $g(x) = 2x$. It's easy to see that $f = g$.

Defining a Function by Cases

Functions can often be defined by cases. For example, the absolute value function "abs" has type $\mathbb{R} \to \mathbb{R}$, and it can be defined by the following rule:

$$\text{abs}(x) = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x < 0. \end{cases}$$

A definition by cases can also be written in terms of the *if-then-else* rule. For example, we can write the preceding definition in the following form:

$$\text{abs}(x) = \text{if } x \geq 0 \text{ then } x \text{ else } -x.$$

The if-then-else rule can be used more than once if there are several cases to define. For example, suppose we want to classify the roots of a quadratic equation having the following form:

$$ax^2 + bx + c = 0.$$

We can define the function "classifyRoots" to give the appropriate statements as follows:

```
classifyRoots($a$, $b$, $c$) = if $b^2 - 4ac > 0$ then
                                   "The roots are real and distinct."
                               else if $b^2 - 4ac < 0$ then
                                   "The roots are complex conjugates."
                               else
                                   "The roots are real and repeated."
```

2.1.2 Some Useful Functions

Now let's look at some functions that are especially useful in computer science. These functions are used for tasks such as analyzing properties of data, analyzing properties of programs, and constructing programs.

The Floor and Ceiling Functions

Let's discuss two important functions that "integerize" real numbers by going down or up to the nearest integer. The *floor* function has type $\mathbb{R} \to \mathbb{Z}$ and is defined by setting floor(x) to the closest integer less than or equal to x. For example, floor(8) = 8, floor(8.9) = 8, and floor(–3.5) = –4. A useful shorthand notation for floor(x) is

$$\lfloor x \rfloor.$$

The *ceiling* function also has type $\mathbb{R} \to \mathbb{Z}$ and is defined by setting ceiling(x) to the closest integer greater than or equal to x. For example, ceiling(8) = 8, ceiling(8.9) = 9, and ceiling(–3.5) = –3. The shorthand notation for ceiling(x) is

$$\lceil x \rceil.$$

Figure 2.3 gives a few sample values for the floor and ceiling functions.

Can you find some relationships between floor and ceiling? For example, is $\lfloor x \rfloor = \lceil x - 1 \rceil$? It's pretty easy to see that if x is an integer, then the statement is false. But if x is not an integer, then there is some integer n such that $n < x < n + 1$ and thus also $n - 1 < x - 1 < n$. In this case it follows that $\lfloor x \rfloor = n = \lceil x - 1 \rceil$. So we can say that $\lfloor x \rfloor = \lceil x - 1 \rceil$ if and only if $x \notin \mathbb{Z}$. This property and some others are listed below. The proofs are similar to the argument we just made.

Floor and Ceiling Properties **(2.1)**

a. $\lfloor x + 1 \rfloor = \lfloor x \rfloor + 1$.

b. $\lceil x - 1 \rceil = \lceil x \rceil - 1$.

c. $\lceil x \rceil = \lfloor x \rfloor$ if and only if $x \in \mathbb{Z}$.

d. $\lceil x \rceil = \lfloor x \rfloor + 1$ if and only if $x \notin \mathbb{Z}$.

e. $\lfloor x \rfloor = \lceil x - 1 \rceil$ if and only if $x \notin \mathbb{Z}$.

x	-2.0	-1.7	-1.3	-1.0	-0.7	-0.3	0.0	0.3	0.7	1.0	1.3	1.7	2.0
$\lfloor x \rfloor$	-2	-2	-2	-1	-1	-1	0	0	0	1	1	1	2
$\lceil x \rceil$	-2	-1	-1	-1	0	0	0	1	1	1	2	2	2

Figure 2.3 Some floor and ceiling values.

Greatest Common Divisor

Let's recall from Section 1.1 that an integer d *divides* an integer n if $d \neq 0$ and there is an integer k such that $n = dk$, and we denote this fact with $d \mid n$. Our focus here will be on the largest of all common divisiors for two integers.

Definiton of Greatest Common Divisor

The *greatest common divisor* of two integers, not both zero, is the largest integer that divides them both. We denote the greatest common divisor of a and b by

$$\gcd(a, b).$$

For example, the common divisors of 12 and 18 are ± 1, ± 2, ± 3, ± 6. So the greatest common divisor of 12 and 18 is 6, and we write $\gcd(12, 18) = 6$. Other examples are $\gcd(-44, -12) = 4$ and $\gcd(5, 0) = 5$. If $a \neq 0$, then $\gcd(a, 0) = |a|$. An important and useful special case occurs when $\gcd(a, b) = 1$. In this case a and b are said to be *relatively prime.* For example, 9 and 4 are relatively prime.

Here are some properties of the greatest common divisor function.

Greatest Common Divisor Properties **(2.2)**

a. $\gcd(a, b) = \gcd(b, a) = \gcd(a, -b)$.

b. $\gcd(a, b) = \gcd(b, a - bq)$ for any integer q.

c. If $g = \gcd(a, b)$, then there are integers x and y such that $g = ax + by$.

d. If $d \mid ab$ and $\gcd(d, a) = 1$, then $d \mid b$.

Property (2.2a) confirms that the ordering of the arguments doesn't matter and that negative numbers have positive greatest common divisors. For example, $\gcd(-4, -6) = \gcd(-4, 6) = \gcd(6, -4) = \gcd(6, 4) = 2$. We'll see shortly how property (2.2b) can help us compute greatest common divisors. Property (2.2c) says that we can write $\gcd(a, b)$ in terms of a and b. For example, $\gcd(15, 9) = 3$, and we can write 3 in terms of 15 and 9 as

$$3 = \gcd(15, 9) = 15(2) + 9(-3).$$

Property (2.2d) is a divisibility property that we'll be using later.

Now let's get down to brass tacks and describe an algorithm to compute the greatest common divisor. Most of us recall from elementary school that we can divide an integer a by a nonzero integer b to obtain two other integers, a quotient q and a remainder r, which satisfy an equation like the following:

$$a = bq + r.$$

For example, if $a = -16$ and $b = 3$, then we can write many equations, each with different values for q and r. For example, the following four equations all have the form $a = bq + r$:

$$\begin{aligned} -16 &= 3 \cdot (-4) + (-4) \\ -16 &= 3 \cdot (-5) + (-1) \\ -16 &= 3 \cdot (-6) + 2 \\ -16 &= 3 \cdot (-7) + 5. \end{aligned}$$

In mathematics and computer science the third equation is by far the most useful. In fact it's a result of a theorem called the *division algorithm*, which we'll state for the record.

Division Algorithm
If a and b are integers and $b \neq 0$, then there are unique integers q and r such that $a = bq + r$, where $0 \leq r < |b|$.

The division algorithm together with property (2.2b) gives us the seeds of an algorithm to compute greatest common divisors. Suppose a and b are integers and $b \neq 0$. The division algorithm gives us the equation $a = bq + r$, where $0 \leq r < |b|$. Solving the equation for r gives $r = a - bq$. This fits the form of (2.2b). So we have the nice equation

$$\gcd(a, b) = \gcd(b, a - bq) = \gcd(b, r).$$

The important point about this equation is that the numbers in $\gcd(b, r)$ are getting closer to zero. Let's see how we can use this equation to compute the greatest common divisor. For example, to compute $\gcd(315, 54)$, we apply the division algorithm to obtain the equation $315 = 54 \cdot 5 + 45$. Thus we know that

$$\gcd(315, 54) = \gcd(54, 45).$$

Now apply the division algorithm again to obtain $54 = 45 \cdot 1 + 9$. So we have

$$\gcd(315, 54) = \gcd(54, 45) = \gcd(45, 9).$$

Continuing, we have $45 = 9 \cdot 5 + 0$, which extends our computation to

$$\gcd(315, 54) = \gcd(54, 45) = \gcd(45, 9) = \gcd(9, 0) = 9.$$

The algorithm that we have been demonstrating is called *Euclid's algorithm*. Since greatest common divisors are always positive, we'll describe the algorithm to calculate $\gcd(a, b)$ for the case in which a and b are natural numbers that are not both zero.

Euclid's Algorithm **(2.3)**

Input natural numbers a and b, not both zero, and output $\gcd(a, b)$.

```
while $b > 0$ do
    Construct $a = bq + r$, where $0 \le r < b$;    (by the division algorithm)
    $a := b$;
    $b := r$
od;
Output $a$.
```

We can use Euclid's algorithm to show how property (2.2c) is satisfied. The idea is to keep track of the equations $a = bq + r$ from each execution of the loop. Then work backwards through the equations to solve for $\gcd(a, b)$ in terms of a and b. For example, in our calculation of gcd(315, 54) we obtained the three equations

$$\begin{aligned} 315 &= 54 \cdot 5 + 45 \\ 54 &= 45 \cdot 1 + 9 \\ 45 &= 9 \cdot 5 + 0. \end{aligned}$$

Starting with the second equation, we can solve for 9. Then we can use the first equation to replace 45. The result is an expression for $9 = \gcd(315, 54)$ written in terms of 315 and 54 as $9 = 315 \cdot (-1) + 54 \cdot 6$.

The Mod Function

If a and b are integers, where $b > 0$, then the division algorithm states that there are two unique integers q and r such that

$$a = bq + r \qquad \text{where} \qquad 0 \le r < b.$$

We say that q is the quotient and r is the remainder upon division of a by b. The remainder $r = a - bq$ is the topic of interest.

Definition of Mod Function

If a and b are integers with $b > 0$, then the remainder upon the division of a by b is denoted

$$a \textit{ mod } b$$

If we agree to fix n as a positive integer, then x mod n takes values in the set $\{0, 1, \ldots, n - 1\}$, which is the set of possible remainders obtained upon division of any integer x by n. For example, each row of the table in Figure 2.4 gives some sample values for x mod n.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x mod 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x mod 2	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
x mod 3	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0
x mod 4	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
x mod 5	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0

Figure 2.4 Sample values of mod function.

We sometimes let $\mathbb{N}_n$ denote the set

$$\mathbb{N}_n = \{0, 1, 2, \ldots, n - 1\}.$$

For example, $\mathbb{N}_0 = \varnothing$, $\mathbb{N}_1 = \{0\}$, and $\mathbb{N}_2 = \{0, 1\}$. So for fixed n, the function f defined by $f(x) = x \bmod n$ has type $\mathbb{Z} \to \mathbb{N}_n$.

A Formula for Mod

Can we find a formula for a mod b in terms of a and b? Sure. We have the following formula

$$a \bmod b = r = a - bq, \qquad \text{where} \qquad 0 \leq r \leq b.$$

So we'll have a formula for a mod b if we can find a formula for the quotient q in terms of a and b. Starting with the inequality $0 \leq r < b$ we have the following sequence of inequalities.

$$\begin{aligned}
0 &\leq r < b. \\
-b &< -r \leq 0, \\
a - b &< a - r \leq a, \\
\frac{a-b}{b} &< \frac{a-r}{b} \leq \frac{a}{b}, \\
\frac{a}{b} - 1 &< \frac{a-r}{b} \leq \frac{a}{b}, \\
\frac{a}{b} - 1 &< q \leq \frac{a}{b}, \qquad \text{since } q = \frac{a-r}{b}.
\end{aligned}$$

Since q is an integer, the last inequality implies that q can be written as the floor expression

$$q = \lfloor a/b \rfloor$$

Since $r = a - bq$, we have the following representation of r when $b > 0$.

$$r = a - b\lfloor a/b \rfloor.$$

This gives us a formula for the mod function.

Formula for Mod Function

$$a \text{ mod } b = a - b\lfloor a/b \rfloor .$$

Properties of Mod

The mod function has many properties. For example, the definition of mod tells us that $0 < x \text{ mod } n < n$ for any integer x. So for any integer x we have

$$(x \text{ mod } n) \text{ mod } n = x \text{ mod } n$$

and

$$x \text{ mod } n = x \text{ iff } 0 \leq x < n.$$

The following list contains some of the most useful properties of the mod function.

Mod Function Properties **(2.4)**

a. $x \text{ mod } n = y \text{ mod } n$ iff n divides $x - y$ iff $(x - y) \text{ mod } n = 0$.

b. $(x + y) \text{ mod } n = ((x \text{ mod } n) + (y \text{ mod } n)) \text{ mod } n$.

c. $(xy) \text{ mod } n = ((x \text{ mod } n)(y \text{ mod } n)) \text{ mod } n$.

d. If $ax \text{ mod } n = ay \text{ mod } n$ and $\gcd(a, n) = 1$, then $x \text{ mod } n = y \text{ mod } n$.

e. If $\gcd(a, n) = 1$, then $1 \text{ mod } n = ax \text{ mod } n$ for some integer x.

Proof: We'll prove parts (a) and (d) and discuss the other properties in the exercises. Using the definition of mod we can write

$$x \text{ mod } n = x - nq_1 \text{ and } y \text{ mod } n = y - nq_2$$

for some integers q_1 and q_2. Now we have a string of iff statements.

$$\begin{aligned} x \text{ mod } n = y \text{ mod } n \quad &\text{iff} \quad x - nq_1 = y - nq_2 \\ &\text{iff} \quad x - y = n(q_1 - q_2) \\ &\text{iff} \quad n \text{ divides } (x - y) \\ &\text{iff} \quad (x - y) \text{ mod } n = 0. \end{aligned}$$

So part (a) is true.

For part (d), assume that $ax \text{ mod } n = ay \text{ mod } n$ and $\gcd(a, n) = 1$. By part (a) we can say that n divides $(ax - ay)$. So n divides the product $a(x - y)$. Since $\gcd(a, n) = 1$, it follows from (2.2d) that n divides $(x - y)$. So again by part (a) we have $x \text{ mod } n = y \text{ mod } n$. QED.

example **2.5 Converting Decimal to Binary**

How can we convert a decimal number to binary? For example, the decimal number 53 has the binary representation 110101. The rightmost bit (binary digit) in this representation of 53 is 1 because 53 is an odd number. In general, we can find the rightmost bit (binary digit) of the binary representation of a natural decimal number x by evaluating the expression $x \bmod 2$. In our example, $53 \bmod 2 = 1$, which is the rightmost bit.

So we can apply the division algorithm, dividing 53 by 2, to obtain the rightmost bit as the remainder. This gives us the equation

$$53 = 2 \cdot 26 + 1.$$

Now do the same thing for the quotient 26 and the succeeding quotients.

$$\begin{aligned} 53 &= 2 \cdot 26 + 1 \\ 26 &= 2 \cdot 13 + 0 \\ 13 &= 2 \cdot 6 + 1 \\ 6 &= 2 \cdot 3 + 0 \\ 3 &= 2 \cdot 1 + 1 \\ 1 &= 2 \cdot 0 + 1 \\ 0. & \quad \text{(done)} \end{aligned}$$

We can read off the remainders in the above equations from bottom to top to obtain the binary representation 110101 for 53. An important point to notice is that we can represent any natural number x in the form

$$x = 2\lfloor x/2 \rfloor + x \bmod 2.$$

So an algorithm to convert x to binary can be implemented with the floor and mod functions.

end example

The Log Function

The "log" function—which is shorthand for logarithm—measures the size of exponents. We start with a positive real number $b \neq 1$. If x is a positive real number, then

$$\log_b x = y \text{ means } b^y = x,$$

and we say, "log base b of x is y."

The base-2 log function $\log_2$ occurs frequently in computer science because many algorithms make binary decisions (two choices) and binary trees are useful data structures. For example, suppose we have a binary search tree with 16 nodes having the structure shown in Figure 2.5.

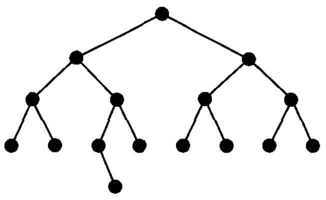


Figure 2.5 Sample binary tree.

x	1	2	4	8	16	32	64	128	256	512	1024
$\log_2 x$	0	1	2	3	4	5	6	7	8	9	10

Figure 2.6 Sample log values.

The depth of the tree is 4, so a maximum of 5 comparisons are needed to find any element in the tree. Notice in this case that $16 = 2^4$, so we can write the depth in terms of the number of nodes: $4 = \log_2 16$. Figure 2.6 gives a few choice values for the $\log_2$ function.

Of course, $\log_2$ takes real values also. For example, if $8 < x < 16$, then

$$3 < \log_2 x < 4.$$

For any real number $b > 1$, the function $\log_b$ is an increasing function with the positive real numbers as its domain and the real numbers as its range. In this case the graph of $\log_b$ has the general form shown in Figure 2.7. What does the graph look like if $0 < b < 1$?

The log function has many properties. For example, it's easy to see that

$$\log_b 1 = 0 \text{ and } \log_b b = 1.$$

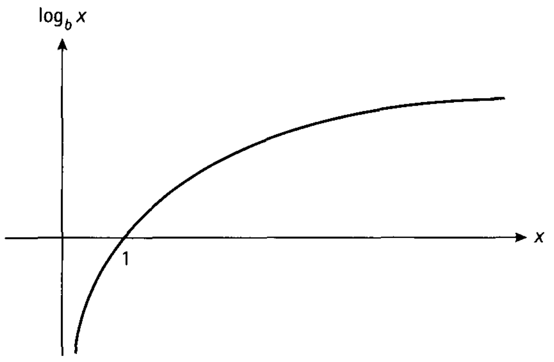


Figure 2.7 Graph of a log function.

The following list contains some of the most useful properties of the log function. We'll leave the proofs as exercises in applying the definition of log.

Log Function Properties **(2.5)**

a. $\log_b (b^x) = x$.

b. $\log_b (x\ y) = \log_b x + \log_b y$.

c. $\log_b(x^y) = y \log_b x$.

d. $\log_b (x/y) = \log_b x - \log_b y$.

e. $\log_a x = (\log_a b)(\log_b x)$. (change of base)

These properties are useful in the evaluation of log expressions. For example, suppose we need to evaluate the expression $\log_2(2^7 3^4)$. Make sure you can justify each step in the following evaluation:

$$\log_2(2^7 3^4) = \log_2(2^7) + \log_2(3^4) = 7 \log_2(2) + 4 \log_2(3) = 7 + 4 \log_2(3).$$

At this point we're stuck for an exact answer. But we can make an estimate. We know that $1 = \log_2(2) < \log_2(3) < \log_2(4) = 2$. Therefore, $1 < \log_2(3) < 2$. Thus we have the following estimate of the answer:

$$11 < \log_2(2^7 3^4) < 15.$$

2.1.3 Partial Functions

A *partial function* from A to B is like a function except that it might not be defined for some elements of A. In other words, some elements of A might not be associated with any element of B. But we still have the requirement that if $x \in A$ is associated with $y \in B$, then x can't be associated with any other element of B. For example, we know that division by zero is not allowed. Therefore, $\div$ is a partial function of type $\mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ because $\div$ is not defined for all pairs of the form $(x, 0)$.

To avoid confusion when discussing partial functions, we use the term *total function* to mean a function that is defined on all its domain. Any partial function can be transformed into a total function. One simple technique is to shrink the domain to the set of elements for which the partial function is defined. For example, $\div$ is a total function of type $\mathbb{R} \times (\mathbb{R} - \{0\}) \rightarrow \mathbb{R}$.

A second technique keeps the domain the same but increases the size of the codomain. For example, suppose $f : A \rightarrow B$ is a partial function. Pick some symbol that is not in B, say $\# \notin B$, and assign $f(x) = \#$ whenever $f(x)$ is not defined. Then we can think of f as a total function of type $A \rightarrow B \cup \{\#\}$. In programming, the analogy would be to pick an error message to indicate that an incorrect input string has been received.

Exercises

Definitions and Examples

1. Describe all possible functions for each of the following types.

   a. $\{a, b\} \to \{1\}$.
   b. $\{a\} \to \{1, 2, 3\}$.
   c. $\{a, b\} \to \{1, 2\}$.
   d. $\{a, b, c\} \to \{1, 2\}$.

2. Suppose we have a function $f : \mathbb{N} \to \mathbb{N}$ defined by $f(x) = 2x + 1$. Describe each of the following sets, where E and O denote the sets of even and odd natural numbers, respectively.

   a. range(f). b. $f(E)$. c. $f(O)$.
   d. $f(\varnothing)$. e. $f^{-1}(E)$. f. $f^{-1}(O)$.

Some Useful Functions

3. Evaluate each of the following expressions.

   a. $\lfloor -4.1 \rfloor$. b. $\lceil -4.1 \rceil$. c. $\lfloor 4.1 \rfloor$ d. $\lceil 4.1 \rceil$.

4. Evaluate each of the following expressions.

   a. gcd(−12, 15). b. gcd(98, 35). c. gcd(872, 45).

5. Find gcd(296, 872) and write the answer in the form $296x + 872y$.

6. Evaluate each of the following expressions.

   a. 15 mod 12. b. −15 mod 12.
   c. -12 mod 15. d. −21 mod 15.

7. Let $f : \mathbb{N}_6 \to \mathbb{N}_6$ be defined by $f(x) = 2x \bmod 6$. Find the image under f of each of the following sets:

   a. $\varnothing$. b. $\{0, 3\}$. c. $\{2, 5\}$.
   d. $\{3, 5\}$. e. $\{1, 2, 3\}$. f. $\mathbb{N}_6$.

8. For a real number x, let trunc(x) denote the truncation of x, which is the integer obtained from x by deleting the part of x to the right of the decimal point.

   a. Write the floor function in terms of trunc.
   b. Write the ceiling function in terms of trunc.

9. For integers x and $y \neq 0$, let $f(x, y) = x - y\ \mathrm{trunc}(x/y)$, where trunc is from Exercise 8. How does f compare to the mod function?

10. Does it make sense to extend the definition of the mod function to real numbers? What would be the range of the function f defined by $f(x) = x \bmod 2.5$?

11. Evaluate each of the following expressions.

    a. $\log_5 625$. b. $\log_2 8192$. c. $\log_3 (1/27)$.

12. For a subset S of a set U, the *characteristic* function $\chi_S : U \to \{0, 1\}$ is a test for membership in S and is defined by

$$\chi_S(x) = \text{if } x \in S \text{ then } 1 \text{ else } 0.$$

    a. Verify that the following equation is correct for subsets A and B of U.

$$\chi_{A \cup B}(x) = \chi_A(x) + \chi_B(x) - \chi_A(x)\chi_B(x)$$

    b. Find a formula for $\chi_{A \cap B}(x)$ in terms of $\chi_A(x)$ and $\chi_B(x)$.
    c. Find a formula for $\chi_{A-B}(x)$ in terms of $\chi_A(x)$ and $\chi_B(x)$.

13. Given a function $f : A \to A$. An element $a \in A$ is called a *fixed point* of f if $f(a) = a$. Find the set of fixed points for each of the following functions.

    a. $f : A \to A$where $f(x) = x$.
    b. $f : \mathbb{N} \to \mathbb{N}$ where $f(x) = x + 1$.
    c. $f : \mathbb{N}_6 \to \mathbb{N}_6$ where $f(x) = 2x \bmod 6$.
    d. $f : \mathbb{N}_6 \to \mathbb{N}_6$ where $f(x) = 3x \bmod 6$.

Proofs and Challenges

14. Prove each of the following statements about floor and ceiling.

    a. $\lfloor x + 1 \rfloor = \lfloor x \rfloor + 1$.
    b. $\lceil x - 1 \rceil = \lceil x \rceil - 1$.
    c. $\lceil x \rceil = \lfloor x \rfloor$ if and only if $x \in \mathbb{Z}$.
    d. $\lceil x \rceil = \lfloor x \rfloor + 1$ if and only if $x \in \mathbb{Z}$.

15. Use the definition of the logarithm function to prove each of the following facts.

    a. $\log_b 1 = 0$.
    b. $\log_b b = 1$.
    c. $\log_b (b^x) = x$.
    d. $\log_b (x\, y) = \log_b x + \log_b y$.
    e. $\log_b(x^y) = y \log_b x$.
    f. $\log_b(x/y) = \log_b x - \log_b y$.
    g. $\log_a x = (\log_a b)\,(\log_b x)$. (change of base)

16. Prove each of the following facts about greatest common divisors.

    a. $\gcd(a, b) = \gcd(b, a) = \gcd(a, -b)$.
    b. $\gcd(a, b) = \gcd(b, a - bq)$ for any integer q.
    c. If $d \mid ab$ and $\gcd(d, a) = 1$, then $d \mid b$. *Hint:* Use (2.2c).

17. Given the result of the division algorithm $a = bq + r$, where $0 \le r < |b|$, prove the following statement:

$$\text{If } b < 0 \text{ then } r = a - b\lceil a/b \rceil.$$

18. Let $f : A \to B$ be a function, and let E and F be subsets of A. Prove each of the following facts about images.

    a. $f(E \cup F) = f(E) \cup f(F)$.
    b. $f(E \cap F) \subset f(E) \cap f(F)$.
    c. Find an example to show that part (b) can be a proper subset.

19. Let $f : A \to B$ be a function, and let G and H be subsets of B. Prove each of the following facts about pre-images.

    a. $f^{-1}(G \cup H) = f^{-1}(G) \cup f^{-1}(H)$.
    b. $f^{-1}(G \cap H) = f^{-1}(G) \cap f^{-1}(H)$.
    c. $E \subset f^{-1}(f(E))$.
    d. $f(f^{-1}(G)) \subset G$.
    e. Find examples to show that parts (c) and (d) can be a proper subsets.

20. Prove each of the following properties of the mod function. *Hint:* Use (2.4a) for parts (a) and (b), and use (2.2c) and parts (a) and (b) for part (c).

    a. $(x + y) \bmod n = ((x \bmod n) + (y \bmod n)) \bmod n$.
    b. $(xy) \bmod n = ((x \bmod n)(y \bmod n)) \bmod n$.
    c. If $\gcd(a, n) = 1$, then there is an integer b such that $1 = ab \bmod n$.

21. We'll start a proof of (2.2c): If $g = \gcd(a, b)$, then there are integers x and y such that $g = ax + by$. Proof: Let $S = \{ax + by \mid x, y \in \mathbb{Z} \text{ and } ax + by > 0\}$ and let d be the smallest number in S. Then there are integers x and y such that $d = ax + by$. The idea is to show that $g = d$. Since $g \mid a$ and $g \mid b$, it follows from (1.1b) that $g \mid d$. So $g \le d$. If we can show that $d \mid a$ and $d \mid b$, then we must conclude that $d = g$ because g is the greatest common divisor of a and b. Finish the proof by showing that $d \mid a$ and $d \mid b$. *Hint:* To show $d \mid a$, write $a = dq + r$, where $0 \le r < d$. Argue that r must be 0.

2.2 Constructing Functions

We often construct a new function by combining other simpler functions in some way. The combining method that we'll discuss in this section is called composition. We'll see that it is a powerful tool to create new functions. We'll also introduce the map function as a useful tool for displaying a list of values of a function. Many programming systems and languages rely on the ideas of this section.

2.2.1 Composition of Functions

Composition of functions is a natural process that we often use without even thinking. For example, the expression $\text{floor}(\log_2(5))$ involves the composition of the two functions floor and $\log_2$. To evaluate the expression, we first evaluate $\log_2(5)$, which is a number between 2 and 3. Then we apply the floor function to this number, obtaining the value 2.

Definition of Composition

The *composition* of two functions f and g is the function denoted by $f \circ g$ and defined by

$$(f \circ g)(x) = f(g(x)).$$

Notice that composition makes sense only for values of x in the domain of g such that $g(x)$ is in the domain of f. So if $g : A \to B$ and $f : C \to D$ and $B \subset C$, then the composition $f \circ g$ makes sense. In other words, for every $x \in A$ it follows that $g(x) \in B$, and since $B \subset C$ it follows that $f(g(x)) \in D$. It also follows that $f \circ g : A \to D$.

For example, we'll consider the floor and $\log_2$ functions. These functions have types

$$\log_2 : \mathbb{R}^+ \to \mathbb{R} \text{ and floor} : \mathbb{R} \to \mathbb{Z},$$

where $\mathbb{R}^+$ denotes the set of positive real numbers. So for any positve real number x, the expression $\log_2(x)$ is a real number and thus $\text{floor}(\log_2(x))$ is an integer. So the composition floor $\circ \log_2$ is defined and

$$\text{floor} \circ \log_2 : \mathbb{R}^+ \to \mathbb{Z}.$$

Composition of functions is associative. In other words, if f, g, and h are functions of the right type such that $(f \circ g) \circ h$ and $f \circ (g \circ h)$ make sense, then

$$(f \circ g) \circ h = f \circ (g \circ h).$$

This is easy to establish by noticing that the two expressions $((f \circ g) \circ h)(x)$ and $(f \circ (g \circ h))(x)$ are equal:

$$((f \circ g) \circ h)(x) = (f \circ g)(h(x)) = f(g(h(x))).$$
$$(f \circ (g \circ h))(x) = f((g \circ h)(x)) = f(g(h(x))).$$

So we can feel free to write the composition of three or more functions without the use of parentheses.

But we should observe that composition is usually not a commutative operation. For example, suppose that f and g are defined by $f(x) = x + 1$ and $g(x) = x^2$. To show that $f \circ g \neq g \circ f$, we only need to find one number x such that $(f \circ g)(x) \neq (g \circ f)(x)$. We'll try $x = 5$ and observe that

$$(f \circ g)(5) = f(g(5)) = f(5^2) = 5^2 + 1 = 26.$$
$$(g \circ f)(5) = g(f(5)) = g(5 + 1) = (5 + 1)^2 = 36.$$

A function that always returns its argument is called an *identity* function. For a set A we sometimes write "id_A" to denote the identity function defined by $\text{id}_A(a) = a$ for all $a \in A$. If $f : A \to B$, then we always have the following equation.

$$f \circ \text{id}_A = f = \text{id}_B \circ f.$$

The Sequence, Distribute, and Pairs Functions

We'll describe here three functions that are quite useful as basic tools to construct more complicated functions that involve lists.

The *sequence* function "seq" has type $\mathbb{N} \to \text{lists}(\mathbb{N})$ and is defined as follows for any natural number n:

$$\text{seq}(n) = \langle 0, 1, \ldots, n \rangle.$$

For example, $\text{seq}(0) = \langle 0 \rangle$ and $\text{seq}(4) = \langle 0, 1, 2, 3, 4 \rangle$.

The *distribute* function "dist" has type $A \times \text{lists}(B) \to \text{lists}(A \times B)$. It takes an element x from A and a list y from $\text{lists}(B)$ and returns the list of pairs made up by pairing x with each element of y. For example,

$$\text{dist}(x, \langle r, s, t \rangle) = \langle (x, r), (x, s), (x, t) \rangle.$$

The *pairs* function takes two lists of equal length and returns the list of pairs of corresponding elements. For example,

$$\text{pairs}(\langle a, b, c \rangle, \langle d, e, f \rangle) = \langle (a, d), (b, e), (c, f) \rangle.$$

Since the domain of pairs is a proper subset of $\text{lists}(A) \times \text{lists}(B)$, it is a partial function of type $\text{lists}(A) \times \text{lists}(B) \to \text{lists}(A \times B)$.

Composing Functions with Different Arities

Composition can also occur between functions with different arities. For example, suppose we define the following function.

$$f(x,\ y) = \text{dist}(x,\ \text{seq}(y)).$$

In this case dist has two arguments and seq has one argument. For example, we'll evaluate the expression $f(5,\ 3)$.

$$\begin{aligned} f(5,3) &= \text{dist}(5, \text{seq}(3)) \\ &= \text{dist}(5, \langle 0,1,2,3 \rangle) \\ &= \langle (5,0),(5,1),(5,2),(5,3) \rangle. \end{aligned}$$

In the next example we'll show that the definition $f(x,\ y) = \text{dist}(x,\ \text{seq}(y))$ is a special case of the following more general form of *composition*, where X can be replaced by any number of arguments.

$$f(X) = h(g_1(X),\ \ldots,\ g_n(X)).$$

example 2.6 Distribute a Sequence

We'll show that the definition $f(x,\ y) = \text{dist}(x,\ \text{seq}(y))$ fits the general form of composition. To make it fit the form, we'll define the functions $\text{one}(x,\ y) = x$ and $\text{two}(x,\ y) = y$. Then we have the following representation of f.

$$\begin{aligned} f(x,y) &= \text{dist}(x, \text{seq}(y)) \\ &= \text{dist}(\text{one}(x,y), \text{seq}(\text{two}(x,y))) \\ &= \text{dist}(\text{one}(x,y), (\text{seq} \circ \text{two})(x,y))). \end{aligned}$$

The last expression has the general form of composition

$$f(X) = h(g_1(X),\ g_2(X)),$$

where $X = (x,\ y)$, $h = \text{dist}$, $g_1 = \text{one}$, and $g_2 = \text{seq} \circ \text{two}$.

end example

example 2.7 The Max Function

Suppose we define the function "max," to return the maximum of two numbers as follows.

$$\max(x,\ y) = \text{if } x < y \text{ then } y \text{ else } x.$$

Then we can use max to define the function "max3," which returns the maximum of three numbers, by the following composition.

$$\text{max3}(x,\ y,\ z) = \max(\max(x,\ y),\ z).$$

end example

We can often construct a function by first writing down an informal definition and then proceeding by stages to transform the definition into a formal one that suits our needs. For example, we might start with an informal definition of some function f such as

$$f(x) = \text{expression involving } x.$$

Now we try to transform the right side of the equality into an expression that has the degree of formality that we need. For example, we might try to reach a composition of known functions as follows:

$$\begin{aligned} f(x) &= \text{expression involving } x \\ &= \text{another expression involving } x \\ &= \dots \\ &= g(h(x)). \end{aligned}$$

From a programming point of view, our goal would be to find an expression that involves known functions that already exist in the programming language being used. Let's do some examples to demonstrate how composition can be useful in solving problems.

example

2.8 Minimum Depth of a Binary Tree

Suppose we want to find the minimum depth of a binary tree in terms of the numbers of nodes. Figure 2.8 lists a few sample cases in which the trees are as compact as possible, which means that they have the least depth for the number of nodes. Let n denote the number of nodes. Notice that when $4 \le n < 8$, the depth is 2. Similarly, the depth is 3 whenever $8 \le n < 16$.

At the same time we know that $\log_2(4) = 2$, $\log_2(8) = 3$, and for $4 \le n < 8$ we have $2 \le \log_2(n) < 3$. So $\log_2(n)$ almost works as the depth function. The problem is that the depth must be exactly 2 whenever $4 \le n < 8$. Can we make this happen? Sure—just apply the floor function to $\log_2(n)$ to get $\text{floor}(\log_2(n)) = 2$ if $4 \le n < 8$. This idea extends to the other intervals that make up $\mathbb{N}$. For example, if $8 \le n < 16$, then $\text{floor}(\log_2(n)) = 3$.

So it makes sense to define our minimum depth function as the composition of the floor function and the $\log_2$ function:

$$\text{minDepth}(n) = \text{floor}(\log_2(n)).$$

end example

example

2.9 A List of Pairs

Suppose we want to construct a definition for the following function in terms of known functions.

$$f(n) = \langle(0, 0), (1, 1), \dots, (n, n)\rangle \qquad \text{for any } n \in \mathbb{N}.$$

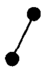
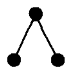
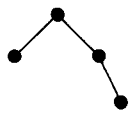
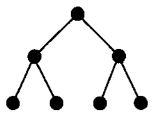
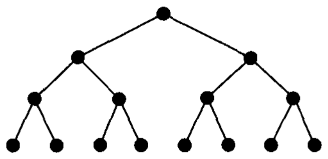
Binary tree	Nodes	Depth
	1	0
	2	1
	3	1
	4	2
	7	2
	15	3

Figure 2.8 Compact binary trees.

Starting with this informal definition, we'll transform it into a composition of known functions.

$$\begin{aligned} f(n) &= \langle (0,0), (1,1), \ldots, (n,n) \rangle \\ &= \text{pairs}(\langle 0,1,\ldots,n\rangle, \langle 0,1,\ldots,n\rangle) \\ &= \text{pairs}(\text{seq}(n), \text{seq}(n)). \end{aligned}$$

Can you figure out the type of f?

end example

example **2.10 Another List of Pairs**

Suppose we want to construct a definition for the following function in terms of known functions.

$$g(k) = \langle (k, 0), (k, 1), \ldots, (k, k) \rangle \qquad \text{for any } k \in \mathbb{N}.$$

Starting with this informal definition, we'll transform it into a composition of known functions.

$$\begin{aligned} g(k) &= \langle (k,0),(k,1),\ldots,(k,k)\rangle \\ &= \text{dist}(k,\langle 0,1,\ldots,k\rangle) \\ &= \text{dist}(k,\text{seq}(k)). \end{aligned}$$

Can you figure out the type of g?

end example

2.2.2 The Map Function

We sometimes need to compute a list of values of a function. A useful tool to accomplish this is the *map* function. It takes a function $f : A \to B$ and a list of elements from A and it returns the list of elements from B constructed by applying f to each element of the given list from A. Here is the definition.

Definition of the Map Function

Let f be a function with domain A and let $\langle x_1, \ldots, x_n\rangle$ be a list of elements from A. Then

$$\text{map}(f, \langle x_1, \ldots, x_n\rangle) = \langle f(x_1), \ldots, f(x_n)\rangle.$$

So the type of the map function can be written as

$$\text{map: } (A \to B) \times \text{lists}(A) \to \text{lists}(B).$$

Here are some example calculations.

$$\begin{aligned} &\text{map}(\text{floor}, \langle -1,5,-0.5,0.5,1.5,2.5\rangle) \\ &\qquad = \langle \text{floor}(-1.5),\text{floor}(-0.5),\text{floor}(0.5),\text{floor}(1.5),\text{floor}(2.5)\rangle \\ &\qquad = \langle -2,-1,0,1,2\rangle. \\ &\text{map}(\text{floor}\circ\log_2, \langle 2,3,4,5\rangle) \\ &\qquad = \langle \text{floor}(\log_2(2)),\text{floor}(\log_2(3)),\text{floor}(\log_2(4)),\text{floor}(\log_2(5))\rangle \\ &\qquad = \langle 1,1,2,2\rangle. \\ &\text{map}(+,\langle (1,2),(3,4),(5,6),(7,8),(9,10)\rangle) \\ &\qquad = \langle +(1,2),+(3,4),+(5,6),+(7,8),+(9,10)\rangle \\ &\qquad = \langle 3,7,11,15,19\rangle. \end{aligned}$$

The map function is an example of a *higher-order* function, which is any function that either has a function as an argument or has a function as a value. This is an important property that most good programming languages possess.

The composition and tupling operations are examples of functions that take other functions as arguments and return functions as results.

example

2.11 A List of Squares

Suppose we want to compute sequences of squares of natural numbers, such as 0, 1, 4, 9, 16. In other words, we want to compute $f : \mathbb{N} \to \text{lists}(\mathbb{N})$ defined by $f(n) = \langle 0, 1, 4, \ldots, n^2 \rangle$. We'll present two solutions. For the first solution we'll define $s(x) = x * x$ and then construct a definition for f in terms of map, s, and seq as follows.

$$\begin{aligned} f(n) &= \langle 0, 1, 4, \ldots, n^2 \rangle \\ &= \langle s(0), s(1), s(2), \ldots, s(n) \rangle \\ &= \text{map}(s, \langle 0, 1, 2, \ldots, n \rangle) \\ &= \text{map}(s, \text{seq}(n)). \end{aligned}$$

For the second solution we'll construct a definition for f without using the function s that we defined for the first solution.

$$\begin{aligned} f(n) &= \langle 0, 1, 4, \ldots, n^2 \rangle \\ &= \langle 0 * 0, 1 * 1, 2 * 2, \ldots, n * n \rangle \\ &= \text{map}(*, \langle (0, 0), (1, 1), 2, 2), \ldots, (n, n) \rangle) \\ &= \text{map}(*, \text{pairs}(\langle 0, 1, 2, \ldots, n \rangle, \langle 0, 1, 2, \ldots, n \rangle)) \\ &= \text{map}(*, \text{pairs}(\text{seq}(n), \text{seq}(n))). \end{aligned}$$

end example

example

2.12 Graphing with Map

Suppose we have a function f defined on the closed interval $[a, b]$ and we have a list of numbers $\langle x_0, \ldots, x_n \rangle$ that form a regular partition of $[a, b]$. We want to find the following sequence of $n + 1$ points:

$$\langle (x_0, f(x_0)), \ldots, (x_n, f(x_n)) \rangle.$$

The partition is defined by $x_i = a + dk$ for $0 \leq k \leq n$, where $d = (b - a)/n$. So the sequence is a function of a, d, and n. If we can somehow create the two lists $\langle x_0, \ldots, x_n \rangle$ and $\langle f(x_0), \ldots, f(x_n) \rangle$, then the desired sequence of points can be obtained by applying the pairs function to these two sequences.

Let "makeSeq" be the function that returns the list $\langle x_0, \ldots, x_n \rangle$. We'll start by trying to define makeSeq in terms of functions that are already at hand. First we write down the desired value of the expression, makeSeq(a, d, n) and then try to gradually transform the value into an expression involving known functions and the arguments a, d, and n.

$$\begin{aligned}
&\text{makeSeq}(a, d, n)\\
&\qquad = \langle x_0, x_1, \ldots, x_n\rangle\\
&\qquad = \langle a, a+d, a+2d, \ldots, a+nd\rangle\\
&\qquad = \text{map}(+, \langle (a,0), (a,d), (a,2d), \ldots, (a,nd)\rangle)\\
&\qquad = \text{map}(+, \text{dist}(a, \langle 0, d, 2d, \ldots, nd\rangle))\\
&\qquad = \text{map}(+, \text{dist}(a, \text{map}(*, \langle (d,0), (d,1), (d,2), \ldots, (d,n)\rangle)))\\
&\qquad = \text{map}(+, \text{dist}(a, \text{map}(*, \text{dist}(d, \langle 0, 1, 2, \ldots, n\rangle))))\\
&\qquad = \text{map}(+, \text{dist}(a, \text{map}(*, \text{dist}(d, \text{seq}(n))))).
\end{aligned}$$

The last expression contains only known functions and the arguments a, d, and n. So we have a definition for makeSeq. Now it's an easy matter to build the second list. Just notice that

$$\begin{aligned}
\langle f(x_1), \ldots, f(x_n)\rangle &= \text{map}(f, \langle x_0, x_1, \ldots, x_n\rangle)\\
&= \text{map}(f, \text{makeSeq}(a, d, n)).
\end{aligned}$$

Now let "makeGraph" be the name of the function that returns the desired sequence of points. Then makeGraph can be written as follows:

$$\begin{aligned}
\text{makeGraph}(f, a, d, n) &= \langle (x_0, f(x_0)), \ldots, (x_n, f(x_n))\rangle\\
&= \text{pairs}(\text{makeSeq}(a, d, n), \text{map}(f, \text{makeSeq}(a, d, n))).
\end{aligned}$$

This gives us a definition of makeGraph in terms of known functions and the variables f, a, d, and n.

end example

From the programming point of view, there are many other interesting ways to combine functions. But they will take us too far afield. The primary purpose now is to get a feel for what a function is and to grasp the idea of building a function from other functions by composition.

Exercises

Composing Functions

1. Evaluate each of the following expressions.
   a. floor($\log_2$ 17).
   b. ceiling($\log_2$ 25).
   c. gcd(14 mod 6, 18 mod 7).
   d. gcd(12, 18) mod 5.
   e. dist(4, seq(3)).
   f. pairs(seq(3), seq(3)).
   g. dist(+, pairs(seq(2), seq(2))).

2. In each case find the compositions $f \circ g$ and $g \circ f$, and find an integer x such that $f(g(x)) \neq g(f(x))$.

   a. $f(x) = \text{ceiling}(x/2)$ and $g(x) = 2x$.
   b. $f(x) = \text{floor}(x/2)$ and $g(x) = 2x + 1$.
   c. $f(x) = \gcd(x, 10)$ and $g(x) = x \bmod 5$.

3. Let $f(x) = x^2$ and $g(x, y) = x + y$. Find compositions that use the functions f and g for each of the following expressions.

   a. $(x + y)^2$. b. $x^2 + y^2$. c. $(x + y + z)^2$. d. $x^2 + y^2 + z^2$.

4. Describe the set of natural numbers x satisfying each equation.

   a. $\text{floor}(\log_2(x)) = 7$.
   b. $\text{ceiling}(\log_2(x)) = 7$.

5. Find a definition for the function max4 that calculates the maximum value of four numbers. Use only composition and the function max that returns the maximum value of two numbers.

6. Find a formula for the number of binary digits in the binary representation of a nonzero natural number x. *Hint:* Notice, for example, that the numbers from 4 through 7 require three binary digits, while the numbers 8 through 15 require five binary digits, and so on.

Composing with the Map Function

7. Evaluate each expression:

   a. $\text{map}(\text{floor} \circ \log_2, \langle 1, 2, 3, \ldots, 16 \rangle)$.
   b. $\text{map}(\text{ceiling} \circ \log_2, \langle 1, 2, 3, \ldots, 16 \rangle)$.

8. Suppose that $f : \mathbb{N} \to \text{lists}(\mathbb{N})$ is defined by $f(n) = \langle 0, 2, 4, 6, \ldots, 2n \rangle$. For example, $f(5) = \langle 0, 2, 4, 6, 8, 10 \rangle$. In each case find a definition for f as a composition of the listed functions.

   a. map, $+$, pairs, seq.
   b. map, $*$, dist, seq.

9. For each of the following functions, construct a definition of the function as a composition of known functions. Assume that all of the variables are natural numbers.

   a. $f(n, k) = \langle n, n + 1, n + 2, \ldots, n + \text{k} \rangle$.
   b. $f(n, k) = \langle 0, k, 2k, 3k, \ldots, nk \rangle$.
   c. $f(n, m) = \langle n, n + 1, n + 2, \ldots, m - 1, m \rangle$, where $n \leq m$.

d. $f(n) = \langle n, n-1, n-2, \ldots, 1, 0\rangle$.
e. $f(n) = \langle (0, n), (1, n-1), \ldots, (n-1, 1), (n, 0)\rangle$.
f. $f(n) = \langle 1, 3, 5, \ldots, 2n+1\rangle$.
g. $f(g, n) = \langle (0, g(0)), (1, g(1)), \ldots, (n, g(n))\rangle$.
h. $f(g, \langle x_1, x_2, \ldots, x_n\rangle) = \langle (x_1, g(x_1)), (x_2, g(x_2)), \ldots, (x_n, g(x_n))\rangle$.
i. $f(g, h, \langle x_1, \ldots, x_n\rangle) = \langle (g(x_1), h(x_1)), \ldots, (g(x_n), h(x_n))\rangle$.

10. We defined $\text{seq}(n) = \langle 0, 1, 2, 3, \ldots, n\rangle$. Suppose we want the sequence to start with the number 1. In other words, for $n > 0$, we want to define a function $f(n) = \langle 1, 2, 3, \ldots, n\rangle$. Find a definition for f as a composition of the functions map, +, dist, and seq.

Proofs

11. Prove each of the following statements.
a. $\text{floor}(\text{ceiling}(x)) = \text{ceiling}(x)$.
b. $\text{ceiling}(\text{floor}(x)) = \text{floor}(x)$.
c. $\text{floor}(\log_2(x)) = \text{floor}(\log_2(\text{floor}(x)))$ for $x \geq 1$.

2.3 Properties Of Functions

Functions that satisfy one or both of two special properties can be very useful in solving a variety of computational problems. One property is that distinct elements map to distinct elements. The other property is that the range is equal to the codomain. We'll discuss these properties in more detail and give some examples where they are useful.

2.3.1 Injections and Surjections

Injective Functions

A function $f : A \to B$ is called *injective* (also *one-to-one*, or an *embedding*) if it maps distinct elements of A to distinct elements of B. Another way to say this is that f is injective if $x \neq y$ implies $f(x) \neq f(y)$. Yet another way to say this is that f is injective if $f(x) = f(y)$ implies $x = y$. An injective function is called an *injection*.

For example, Figure 2.9 illustrates an injection from a set A to a B.

Surjective Functions

A function $f : A \to B$ is called *surjective* (also *onto*) if the range of f is the codomain B. Another way to say this is that f is surjective if each element

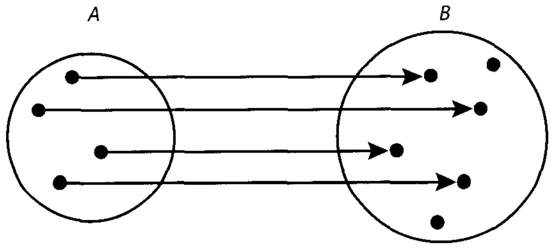


Figure 2.9 An injection.

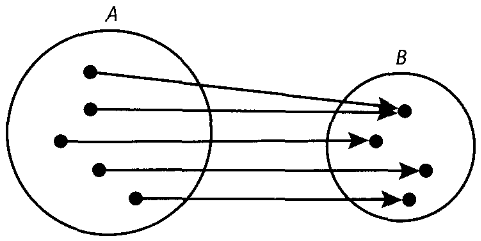


Figure 2.10 A surjection.

$b \in B$ can be written as $b = f(x)$ for some element $x \in A$. A surjective function is called a *surjection*.

For example, Figure 2.10 pictures a surjection from A to B.

example 2.13 Injective or Surjective

We'll give a few examples of functions that have one or the other of the injective and surjective properties.

1. The function $f : \mathbb{R} \to \mathbb{Z}$ defined by $f(x) = \lceil x + 1 \rceil$ is surjective because for any $y \in \mathbb{Z}$ there is a number in $\mathbb{R}$, namely $y - 1$, such that $f(y - 1) = y$. But f is not injective because, for example, $f(3.5) = f(3)$.

2. The function $f : \mathbb{N}_8 \to \mathbb{N}_8$ defined by $f(x) = 2x \bmod 8$ is not injective because, for example, $f(0) = f(4)$. f is not surjective because the range of f is only the set $\{0, 2, 4, 6\}$.

3. Let $g : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ be defined by $g(x) = (x, x)$. Then g is injective because if $x, y \in \mathbb{N}$ and $x \neq y$, then $g(x) = (x, x) \neq (y, y) = g(y)$. But g is not surjective because, for example, nothing maps to $(0, 1)$.

4. The function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ defined by $f(x, y) = 2x + y$ is surjective. To see this, notice that any $z \in \mathbb{N}$ is either even or odd. If z is even, then $z = 2k$ for some $k \in \mathbb{N}$, so $f(k, 0) = z$. If z is odd, then $z = 2k + 1$ for some

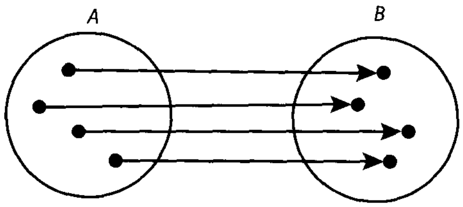


Figure 2.11 A bijection.

$k \in \mathbb{N}$, so $f(k, 1) = z$. Thus f is surjective. But f is not injective because, for example, $f(0, 2) = f(1, 0)$.

end example

2.3.2 Bijections and Inverses

Bijections

A function is called *bijective* if it is both injective and surjective. Another term for bijective is "one-to-one and onto." A bijective function is called a *bijection* or a "one-to-one correspondence."

For example, Figure 2.11 pictures a bijection from A to B.

example 2.14 A Bijection

Let $(0, 1) = \{x \in \mathbb{R} \mid 0 < x < 1\}$ and let $\mathbb{R}^+$ denote the set of positive real numbers. We'll show that the function $f : (0, 1) \to \mathbb{R}^+$ defined by

$$f(x) = \frac{x}{1-x}$$

is a bijection. To show that f is an injection, let $f(x) = f(y)$. Then

$$\frac{x}{1-x} = \frac{y}{1-y},$$

which can be cross multiplied to get $x - xy = y - xy$. Subtract $-xy$ from both sides to get $x = y$. Thus f is injective. To show that f is surjective, let $y > 0$ and try to find $x \in (0, 1)$ such that $f(x) = y$. We'll solve the equation

$$\frac{x}{1-x} = y.$$

Cross multiply and solve for x to obtain

$$x = \frac{y}{y+1}.$$

It follows that $f(y/(y+1)) = y$, and since $y > 0$, it follows that $0 < y/(y+1) < 1$. Thus f is surjective. Therefore, f is a bijection.

end example

Inverse Functions

Bijections always come in pairs. If $f : A \to B$ is a bijection, then there is a function $g : B \to A$, called the *inverse* of f, defined by $g(b) = a$ if $f(a) = b$. Of course, the inverse of f is also a bijection and we have $g(f(a)) = a$ for all $a \in A$ and $f(g(b)) = b$ for all $b \in B$. In other words, $g \circ f = \text{id}_A$ and $f \circ g = \text{id}_B$.

We should note that there is exactly one inverse of any bijection f. Suppose that g and h are two inverses of f. Then for any $x \in B$ we have

$$\begin{aligned} g(x) &= g(\text{id}_B(x)) \\ &= g(f(h(x))) && (\text{since } f \circ h = \text{id}_B) \\ &= \text{id}_A\,(h(x)) && (\text{since } g \circ f = \text{id}_A) \\ &= h(x). \end{aligned}$$

This tells us that $g = h$. The inverse of f is often denoted by the symbol f^{-1}. So if f is a bijection and $f(a) = b$, then $f^{-1}(b) = a$. Notice the close relationship between the equation $f^{-1}(b) = a$ and the pre-image equation $f^{-1}(\{b\}) = \{a\}$.

example **2.15 Inverses**

We'll look at two bijective functions together with their inverses.

1. Let *Odd* and *Even* be the sets of odd and even natural numbers, respectively. The function $f : \textit{Odd} \to \textit{Even}$ defined by $f(x) = x - 1$ is a bijection. Check it out. The inverse of f can be defined by $f^{-1}(x) = x + 1$. Notice that $f^{-1}(f(x)) = f^{-1}(x - 1) = (x - 1) + 1 = x$.

2. The function $f : \mathbb{N}_5 \to \mathbb{N}_5$ defined by $f(x) = 2x$ mod 5 is bijective because, $f(0) = 0$, $f(1) = 2$, $f(2) = 4$, $f(3) = 1$, and $f(4) = 3$. The inverse of f can be defined by $f^{-1}(x) = 3x$ mod 5. For example, $f^{-1}(f(4)) = 3f(4)$ mod 5 = 9 mod 5 = 4. Check the other values too.

end example

The fact that the function $f : \mathbb{N}_5 \to \mathbb{N}_5$ defined by $f(x) = 2x$ mod 5 (in the preceding example) is a bijection is an instance of an interesting and useful fact about the mod function and inverses. Here is the result.

The Mod Function and Inverses (2.6)

Let $n > 1$ and let $f : \mathbb{N}_n \to \mathbb{N}_n$ be defined as follows, where a and b are integers.

$$f(x) = (ax + b) \text{ mod } n.$$

Continued ➡

➧ ➧

Then f is a bijection if and only if $\gcd(a, n) = 1$. When this is the case, the inverse function f^{-1} is defined by

$$f^{-1}(x) = (kx + c) \bmod n,$$

where c is an integer such that $f(c) = 0$, and k is an integer such that $1 = ak + nm$ for some integer m.

Proof: We'll prove the iff part of the statement and leave the form of the inverse as an exercise.

Assume that f is a bijection and show that $\gcd(a, n) = 1$. Then f is surjective, so there are numbers $s, c \in \mathbb{N}_n$ such that $f(s) = 1$ and $f(c) = 0$. Using the definition of f, these equations become

$$(as + b) \bmod n = 1 \text{ and } (ac + b) \bmod n = 0.$$

Therefore, there are intgers q_1 and q_2 such that the two equations become

$$as + b + nq_1 = 1 \text{ and } ac + b + nq_2 = 0.$$

Solve the second equation for b to get $b = -ac - nq_2$, and substitute for b in the first equation to get

$$1 = a(s - c) + n(q_1 - nq_2).$$

Since $\gcd(a, n)$ divides both a and n, it divides the right side of the equation (1.1b) and therefore must also divide 1. Therefore, $\gcd(a, n) = 1$.

Now assume that $\gcd(a, n) = 1$ and show that f is a bijection. Since $\mathbb{N}_n$ is finite, we need only show that f is an injection to conclude that it is a bijection. So let $x, y \in \mathbb{N}_n$ and let $f(x) = f(y)$. Then

$$(ax + b) \bmod n = (ay + b) \bmod n,$$

which by (2.4a) implies that n divides $(ax + b) - (ay + b)$. Therefore, n divides $a(x - y)$, and since $\gcd(a, n) = 1$, we conclude from (2.2d) that n divides $x - y$. But the only way for n to divide $x - y$ is for $x - y = 0$ because both $x, y \in \mathbb{N}_n$. Thus $x = y$, and it follows that f is injective, hence also surjective, and therefore bijective. QED.

Relationships

An interesting property that relates injections and surjections is that if there is an injection from A to B, then there is a surjection from B to A, and conversely. A less surprising fact is that if the composition $f \circ g$ makes sense and if both f and g have one of the properties injective, surjective, or bijective, then $f \circ g$ also has the property. We'll list these facts for the record.

Injective and Surjective Relationships **(2.7)**

a. If f and g are injective, then $g \circ f$ is injective.

b. If f and g are surjective, then $g \circ f$ is surjective.

c. If f and g are bijective, then $g \circ f$ is bijective.

d. There is an injection from A to B if and only if there is a surjection from B to A.

Proof: We'll prove (2.7d) and leave the others as exercises. Suppose that f is an injection from A to B. We'll define a function g from B to A. Since f is an injection, it follows that for each $b \in \text{range}(f)$ there is exactly one $a \in A$ such that $b = f(a)$. In this case, we define $g(b) = a$. For each $b \in B - \text{range}(f)$ we have the freedom to let g map b to any element of A that we like. So g is a function from B to A and we defined g so that $\text{range}(g) = A$. Thus g is surjective.

For the other direction, assume that g is a surjection from B to A. We'll define a function f from A to B. Since g is a surjection, it follows that for each $a \in A$, the pre-image $g^{-1}(\{a\}) \neq \varnothing$. So we can pick an element $b \in g^{-1}(\{a\})$ and define $f(a) = b$. Thus f is a function from A to B. Now if $x,\ y \in A$ and $x \neq y$, then $g^{-1}(\{x\}) \cap g^{-1}(\{y\}) = \varnothing$. Since $f(x) \in g^{-1}(\{x\})$ and $f(y) \in g^{-1}(\{y\})$, it follows that $f(x) \neq f(y)$. Thus f is injective. QED.

2.3.3 The Pigeonhole Principle

We're going to describe a useful rule that we often use without thinking. For example, suppose 21 pieces of mail are placed in 20 mail boxes. Then one mailbox receives at least two pieces of mail. This is an example of the *pigeonhole principle*, where we think of the pieces of mail as pigeons and the mail boxes as pigeonholes.

Pigeonhole Principle

If m pigeons fly into to n pigeonholes where $m > n$, then one pigeonhole will have two or more pigeons.

We can describe the pigeonhole principle in more formal terms as follows: If A and B are finite sets with $|A| > |B|$, then every function from A to B maps at least two elements of A to a single element of B. This is the same as saying that no function from A to B is an injection.

This simple idea is used often in many different settings. We'll be using it at several places in the book.

example **2.16 Pigeonhole Examples**

Here are a few sample statements that can be justified by the pigeonhole principle.

1. The "musical chairs" game is played with n people and $n - 1$ chairs for them to sit on when the music stops.
2. In a group of eight people, two were born on the same day of the week.
3. If a six-sided die is tossed seven times, one side will come up twice.
4. If a directed graph with n vertices has a path of length n or longer, then the path must pass through some vertex at least twice. This implies that the graph contains a cycle.
5. In any set of $n + 1$ integers, there are two numbers that have the same remainder on division by n. This follows because there are only n remainders possible on division by n.
6. The decimal expansion of any rational number contains a repeating sequence of digits (they might be all zeros). For example, $359/495 = 0.7252525\ldots$, $7/3 = 2.333\ldots$, and $2/5 = 0.4000\ldots$. To see this, let m/n be a rational number. Divide m by n until all the digits of m are used up. This gets us to the decimal point. Now continue the division by n for $n + 1$ more steps. This gives us $n + 1$ remainders. Since there are only n remainders possible on division by n, the pigeonhole principle tells us that one of remainders will be repeated. So the sequence of remainders between the repeated remainders will be repeated forever. This causes the corresponding sequence of digits in the decimal expansion to be repeated forever.

end example

2.3.4 Simple Ciphers

Bijections and inverse functions play an important role when working with systems (called ciphers) to encipher and decipher information. We'll give a few examples to illustrate the connections. For ease of discussion we'll denote the 26 letters of the lowercase alphabet by the set $\mathbb{N}_{26} = \{0, 1, 2, \ldots, 25\}$, where we identify a with 0, b with 1, and so on.

To get things started we'll describe a cipher to transform a string of text by means of a simple translation of the characters. For example, the message 'abc' translated by 5 letters becomes 'fgh'. The cipher is easy to write once we figure out how wrap around the end of the alphabet. For example, to translate the letter z (i.e., 25) by 5 letters we need to come up with the letter e (i.e., 4). All we need to do is add the two numbers mod 26:

$$(25 + 5) \bmod 26 = 4.$$

So we can define a cipher f to translate any letter by 5 letters as

$$f(x) = (x + 5) \bmod 26.$$

Is f a bijection? Yes, because f has type $\mathbb{N}_{26} \to \mathbb{N}_{26}$. So we have a cipher for transforming letters. For example, the message 'hello' transforms to 'mjqqt'. To decode the message we need to reverse the process, which is easy in this case because the inverse of f is easy to guess.

$$f^{-1}(x) = (x - 5) \bmod 26.$$

For example, to see that e maps back to z, we can observe that 4 maps to 25.

$$f^{-1}(4) = (4 - 5) \bmod 26 = (-1) \bmod 26 = 25.$$

The cipher we've been talking about is called an *additive* cipher. A cryptanalyst who intercepts the message 'mjqqt' can easily check whether it was created by an additive cipher. An additive cipher is an example of a *monoalphabetic* cipher, which is a cipher that always replaces any character of the alphabet by the same character from the cipher alphabet.

A *multiplicative* cipher is a monoalphabetic cipher that translates each letter by using a multiplier. For example, suppose we define the cipher

$$g(x) = 3x \bmod 26.$$

For example, this cipher maps a to a, c to g, and m to k. Is g a bijection? You can convince yourself that it is by exaustive checking. But it's easier to use (2.6). Since $\gcd(3, 26) = 1$ it follows that g is a bijection. What abut deciphering? Again, (2.6) comes to the rescue to tell us the form of g^{-1}. Since we can write $\gcd(3, 26) = 1 = 3(9) + 26(-1)$, and since $g(0) = 0$, it follows that we can define g^{-1} as

$$g^{-1}(x) = 9x \bmod 26.$$

There are some questions to ask about multiplicative ciphers. Which keys act as an identity (not changing the message)? Is there always one letter that never changes no matter what the key? Do fractions work as keys? What about decoding (i.e., deciphering) a message? Do you need a new deciphering algorithm?

An *affine* cipher is a monoalphabetic cipher that translates each letter by using two kinds of translation. For example, we can start with a pair of keys (M, A) and transform a letter by first applying the additive cipher with key A to get an intermediate letter. Then apply the multiplicative cipher with key M to that letter to obtain the desired letter. For example, we might use the pair of keys (5, 3) and define f as

$$f(x) = 3((x + 5) \bmod 26) \bmod 26 = (3x + 5) \bmod 26.$$

We can use (2.6) to conclude that f is a bijection because gcd(3, 26) = 1. So we can also decipher messages with f^{-1}, which we can construct using (2.6) as

$$f^{-1}(x) = (9x + 7) \bmod 26.$$

Some ciphers leave one or more letters fixed. For example, an additive cipher that translates by a multiple of 26 will leave all letters fixed. A multiplicative cipher always sends 0 to 0, so one letter is fixed. But what about an affine cipher of the form $f(x) = (ax + b) \bmod 26$? When can we be sure that no letters are fixed? In other words, when can we be sure that $f(x) \neq x$ for all $x \in N_{26}$? The answer is, when $\gcd(a - 1, 26)$ does not divide b. Here is the general result that we've been discussing.

The Mod Function and Fixed Points **(2.8)**

Let $n > 1$ and let $f : \mathbb{N}_n \to \mathbb{N}_n$ be defined as follows, where a and b are integers.

$$f(x) = (ax + b) \bmod n.$$

Then f has no fixed points (i.e., f changes every letter of an alphabet) if and only if $\gcd(a - 1, n)$ does not divide b.

This result follows from an old and easy result from number theory, and we'll discuss it in the exercises. Let's see how the result helps our cipher problem.

example **2.17 Simple Ciphers**

The function $f(x) = (3x + 5) \bmod 26$ does not have any fixed points because $\gcd(3 - 1, 26) = \gcd(2, 26) = 2$, and 2 does not divide 5. It's nice to know that we don't have to check all 26 values of f.

On the other hand, the function $f(x) = (3x + 4) \bmod 26$ has fixed points because $\gcd(3 - 1, 26) = 2$ and 2 divides 4. For this example, we can observe that $f(11) = 11$ and $f(24) = 24$. So in terms of our association of letters with numbers we would have $f(l) = l$ and $f(y) = y$.

end example

Whatever cipher we use, we always have some questions: Is it a bijection? What is the range of values for the keys? Is it hard to decipher an intercepted message?

2.3.5 Hash Functions

Suppose we wish to retrieve some information stored in a table of size n with indexes $0, 1, \ldots, n-1$. The items in the table can be very general things. For example, the items might be strings of letters, or they might be large records with many fields of information. To look up a table item we need a *key* to the information we desire.

For example, if the table contains records of information for the 12 months of the year, the keys might be the three-letter abbreviations for the 12 months. To look up the record for January, we would present the key Jan to a lookup program. The program uses the key to find the table entry for the January record of information. Then the information would be available to us.

An easy way to look up the January record is to search the table until the key Jan is found. This might be OK for a small table with 12 entries. But it may be impossibly slow for large tables with thousands of entries. Here is the general problem that we want to solve:

Given a key, find the table entry containing the key without searching.

This may seem impossible at first glance. But let's consider a way to use a function to map each key directly to its table location.

Definition of Hash Function

A *hash function* is a function that maps a set S of keys to a finite set of table indexes, which we'll assume are $0, 1, \ldots, n-1$. A table whose information is found by a hash function is called a *hash table.*

For example, let S be the set of three-letter abreviations for the months of the year. We might define a hash function $f : S \to \{0, 1, \ldots, 11\}$ in the following way.

$$f(XYZ) = (\text{ord}(X) + \text{ord}(Y) + \text{ord}(Z)) \bmod 12.$$

where ord(X) denotes the integer value of the ASCII code for X. (The ASCII values for A to Z and a to z are 65 to 90 and 97 to 122, respectively.) For example, we'll compute the value for the key Jan.

$$\begin{aligned} f(\text{Jan}) &= (\text{ord}(J) + \text{ord}(a) + \text{ord}(n)) \mod 12 \\ &= (74 + 97 + 110) \mod 12 \\ &= 5. \end{aligned}$$

Most programming languages have efficient implementations of the ord and mod functions, so hash functions constructed from them are quite fast. Here is the listing of all the values of f.

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
5	5	0	3	7	1	11	9	8	6	7	4

Notice the function f is not injective. For example, $f(\text{Jan}) = f(\text{Feb}) = 5$. So if we use f to construct a hash table, we can't put the information for January and February at the same address. Let's discuss this problem.

Collisions

If a hash function is injective, then it maps every key to the index of the hash table where the information is stored and no searching is involved. Often this is not possible. When two keys map to the same table index, the result is called a *collision.* So if a hash function is not injective, it has collisions. Our example hash function has collisions $f(\text{Jan}) = f(\text{Feb})$ and $f(\text{May}) = f(\text{Nov})$.

When collisions occur, we store the information for one of the keys in the common table location and must find some other location for the other keys. There are many ways to find the location for a key that has collided with another key. One technique is called *linear probing.* With this technique the program searches the remaining locations in a "linear" manner.

For example, if location k is the collision index, then the following sequence of table locations is searched

$$(k + 1) \bmod n,\ (k + 2) \bmod n,\ \ldots,\ (k + n) \bmod n.$$

In constructing the table in the first place, these locations would be searched to find the first open table entry. Then the key would be placed in that location.

example

2.18 A Hash Table

We'll use the sample hash function f to construct a hash table for the months of the year by placing the three-letter abreviations in the table one by one, starting with Jan and continuing to Dec. We'll use linear probing to resolve collisions that occur in the process. For example, since $f(\text{Jan}) = 5$, we place Jan in position 5 of the table. Next, since $f(\text{Feb}) = 5$ and since postition 5 is full, we look for the next available position and place Feb in postition 6. Continuing in this way, we eventually construct the following hash table, where entries in parentheses need some searching to be found.

0	1	2	3	4	5	6	7	8	9	10	11
Mar	Jun	(Nov)	Apr	Dec	Jan	(Feb)	May	Sep	Aug	(Oct)	Jul

There are many questions. Can we find an injection so there are no collisions? If we increased the size of the table, would it give us a better chance of finding an injection? If the table size is increased, can we scatter the elements so that collisions can be searched for in less time?

end example

Probe Sequences

Linear probing that looks at locations one step at a time may not be the best way to resolve collisions for some kinds of keys. An alternative is to try linear probing with a "gap" between table locations in order to "scatter" or "hash" the information to different parts of the table. The idea is to keep the number of searches to a minimum. Let g be a gap, where $1 \leq g < n$. Then the following sequence of table locations is searched in case a collision occurs at location k:

$$(k + g) \bmod n,\ (k + 2g) \bmod n,\ \ldots,\ (k + ng) \bmod n.$$

Some problems can occur if we're not careful with our choice of g. For example, suppose $n = 12$ and $g = 4$. Then the probe sequence can skip some table entries. For example, if $k = 7$, the above sequence becomes

$$11, 3, 7, 11, 3, 7, 11, 3, 7, 11, 3, 7.$$

So we would miss table entries 0, 1, 2, 4, 5, 6, 8, 9, and 10. Let's try another value for g. Suppose we try $g = 5$. Then we obtain the following probe sequence starting at $k = 7$:

$$0, 5, 10, 3, 8, 1, 6, 11, 4, 9, 2, 7.$$

In this case we cover the entire set $\{0, 1, \ldots, 11\}$. In other words, we've defined a bijection $f : \mathbb{N}_{12} \rightarrow \mathbb{N}_{12}$ by $f(x) = 5x \bmod 12$. Can we always find a probe sequence that hits all the elements of $\{0, 1, \ldots, n - 1\}$? Happily, the answer is yes. Just pick g and n so that they are relatively prime, $\gcd(g, n) = 1$. For example, if we pick n to be a prime number, then $(g, n) = 1$ for any g in the interval $1 \leq g < n$. That's why table sizes are often prime numbers, even though the data set may have fewer entries than the table size.

There are many ways to define hash functions and to resolve collisions. The paper by Cichelli [1980] examines some bijective hash functions.

Exercises

Injections, Surjections, and Bijections

1. The fatherOf function from *People* to *People* is neither injective nor surjective. Why?

2. For each of the following cases, construct a function satisfying the given condition, where the domain and codomain are chosen from the sets

$$A = \{a, b, c\},\ B = \{x, y, z\},\ C = \{1, 2\}.$$

   a. Injective but not surjective.
   b. Surjective but not injective.
   c. Bijective.
   d. Neither injective nor surjective.

3. For each of the following types, compile some statistics: the number of functions of that type; the number that are injective; the number that are surjective; the number that are bijective; the number that are neither injective, surjective, nor bijective.

   a. $\{a, b, c\} \to \{1, 2\}$.
   b. $\{a, b\} \to \{1, 2, 3\}$.
   c. $\{a, b, c\} \to \{1, 2, 3\}$.

4. Show that each function $f : \mathbb{N} \to \mathbb{N}$ has the listed properties.

   a. $f(x) = 2x$. (injective and not surjective)
   b. $f(x) = x + 1$. (injective and not surjective)
   c. $f(x) = \text{floor}(x/2)$. (surjective and not injective)
   d. $f(x) = \text{ceiling}(\log_2 (x + 1))$. (surjective and not injective)
   e. $f(x) =$ if x is odd then $x - 1$ else $x + 1$. (bijective)

5. For each of the following functions, state which of the properties injective and surjective holds.

   a. $f : \mathbb{R} \to \mathbb{Z}$, where $f(x) = \text{floor}(x)$.
   b. $f : \mathbb{N} \to \mathbb{N}$, where $f(x) = x \bmod 10$.
   c. $f : \mathbb{Z} \to \mathbb{N}$ defined by $f(x) = |x + 1|$.
   d. $\text{seq} : \mathbb{N} \to \text{lists}(\mathbb{N})$.
   e. $\text{dist} : A \times \text{lists}(B) \to \text{lists}(A \times B)$.
   f. $f : A \to \text{power}(A)$, A is any set, and $f(x) = \{x\}$.
   g. $f : \text{lists}(A) \to \text{power}(A)$, A is finite, and $f(\langle x_1, \ldots, x_n\rangle) = \{x_1, \ldots, x_n\}$.
   h. $f : \text{lists}(A) \to \text{bags}(A)$, A is finite, and $f(\langle x_1, \ldots, x_n\rangle) = [x_1, \ldots, x_n]$.
   i. $f : \text{bags}(A) \to \text{power}(A)$, A is finite, and $f([x_1, \ldots, x_n]) = \{x_1, \ldots, x_n\}$.

6. Let $\mathbb{R}^+$ and $\mathbb{R}^-$ denote the sets of positive and negative real numbers, respectively. If $a, b \in \mathbb{R}$ and $a < b$, let $(a, b) = \{x \in \mathbb{R} \mid a < x < b\}$. Show that each of the following functions is a bijection.

   a. $f : (0, 1) \to (a, b)$ defined by $f(x) = (b - a)x + a$.
   b. $f : \mathbb{R}^+ \to (0, 1)$ defined by $f(x) = 1/(x + 1)$.
   c. $f : (1/2, 1) \to \mathbb{R}^+$ defined by $f(x) = 1/(2x - 1) - 1$.
   d. $f : (0, 1/2) \to \mathbb{R}^-$ defined by $f(x) = 1/(2x - 1) + 1$.
   e. $f : (0, 1) \to \mathbb{R}$ defined by $f(x) = \begin{cases} 1/(2x-1) - 1 & \text{if } 1/2 < x < 1 \\ 0 & \text{if } x = 1/2 \\ 1/(2x-1) + 1 & \text{if } 0 < x < 1/2 \end{cases}$.

The Pigeonhole Principle

7. Use the pigeonhole principle for each of the following statements.
   a. How many people are needed in a group to say that three were born on the same day of the week?
   b. How many people are needed in a group to say that four were born in the same month?
   c. Why does any set of 10 nonempty strings over $\{a, b, c\}$ contain two different strings whose starting letters agree and whose ending letters agree?
   d. Find the size needed for a set of nonempty strings over $\{a, b, c, d\}$ to contain two strings whose starting letters agree and whose ending letters agree.

8. Use the pigeonhole principle to verify each of the following statements.
   a. In any set of 11 natural numbers, there are two numbers whose decimal representations contain a common digit.
   b. In any set of four numbers picked from the set $\{1, 2, 3, 4, 5, 6\}$, there are two numbers whose sum is seven.
   c. If five distinct numbers are chosen from the set $\{1, 2, 3, 4, 5, 6, 7, 8\}$, then two of the numbers chosen are consecutive (i.e., of the form n and $n + 1$). *Hint:* List the five numbers chosen as, x_1, x_2, x_3, x_4, x_5 and list the successors as $x_1 + 1$, $x_2 + 1$, $x_3 + 1$, $x_4 + 1$, $x_5 + 1$. Are there more than eight numbers listed?

Simple Ciphers and the Mod Function

9. Each of the following functions has the form $f(x) = (ax + b) \text{ mod } n$. Assume that each function has type $\mathbb{N}_n \to \mathbb{N}_n$, so that we can think of f as a cipher for an alphabet represented by the numbers $0, 1, \ldots, n - 1$. Use (2.6) to determine whether each function is a bijection, and, if so, construct its inverse. Then use (2.8) to determine whether the function has fixed points (i.e., letters that don't change), and, if so, find them.
   a. $f(x) = 2x \text{ mod } 6$.
   b. $f(x) = 2x \text{ mod } 5$.
   c. $f(x) = 5x \text{ mod } 6$.
   d. $f(x) = (3x + 2) \text{ mod } 6$.
   e. $f(x) = (2x + 3) \text{ mod } 7$.
   f. $f(x) = (5x + 3) \text{ mod } 12$.
   g. $f(x) = (25x + 7) \text{ mod } 16$.

10. Think of the letters A to Z as the numbers numbers 0 to 25 and let f be a cipher of the form $f(x) = (ax + b) \text{ mod } 26$.

a. Use (2.6) to find all values of a $(0 \leq a < 26)$ that will make f bijective.
b. For the values of a in part (a) that make f bijective, use (2.8) to find a general statement about the values of b $(0 \leq b < 26)$ that will ensure that f maps each letter to a different letter.

Hash Functions

11. Let $S = \{$one, two, three, four, five, six, seven, eight, nine$\}$ and let $f : S \to \mathbb{N}_9$ be defined by $f(x) = (3|x|) \bmod 9$, where $|x|$ means the number of letters in x. For each of the following gaps, construct a hash table that contains the strings of S by choosing a string for entry in the table by the order that it is listed in S. Resolve collisions by linear probing with the given gap and observe whether all strings can be placed in the table.

    a. Gap = 1. b. Gap = 2. c. Gap = 3.

12. Repeat Exercise 11 for the set $S = \{$Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday$\}$ and the function f: $S \to \mathbb{N}_7$ defined by $f(x) = (2|\mathrm{x}| + 3) \bmod 7$.

13. Repeat Exercise 11 for the set $S = \{$January, February, March, April, May, June, July, August$\}$ and f: $S \to \mathbb{N}_7$ defined by $f(x) = (|x| + 3) \bmod 8$.

Proofs and Challenges

14. Find integers a and b such that the function $f : \mathbb{N}_{12} \to \mathbb{N}_{12}$ defined by $f(x) = (ax + b) \bmod 12$ is bijective and $f^{-1} = f$.

15. Let $f : A \to B$ and $g : B \to C$. Prove each of the following statements.

    a. If f and g are injective, then $g \circ f$ is injective.
    b. If f and g are surjective, then $g \circ f$ is surjective.
    c. If f and g are bijective, then $g \circ f$ is bijective.

16. Let f and g be bijections of type $A \to A$ such that $g(f(x)) = x$ for all $x \in A$. Prove that $f(g(x)) = x$ for all $x \in A$.

17. Assume that the functions f and g can be formed into a composition $g \circ f$.

    a. If $g \circ f$ is surjective, what can you say about f or g ?
    b. If $g \circ f$ is injective, what can you say about f or g ?

18. Let $g : A \to B$ and $h : A \to C$ and let f be defined by $f(x) = (g(x), h(x))$. Show that each of the following statements holds.

    a. If f is surjective, then g and h are surjective. Find an example to show that the converse is false.
    b. If g or h is injective, then f is injective. Find an example to show that the converse is false.

19. Prove that the equation $ax \bmod n = b \bmod n$ has a solution x if and only if $\gcd(a, n)$ divides b.

20. Use the result of Exercise 19 to prove (2.8): Let $n > 1$ and $f : \mathbb{N}_n \to \mathbb{N}_n$ be defined by $f(x) = (ax + b) \bmod n$. Then f has no fixed points if and only if $\gcd(a - 1, n)$ does not divide b.

21. Prove the second part of (2.6). In other words, assume the following facts.

    $f : \mathbb{N}_n \to \mathbb{N}_n$ is defined by $f(x) = (ax + b) \bmod n$.
    f is a bijection, which also means that $\gcd(a, n) = 1$.
    c is an integer such that $f(c) = 0$.
    k is an integer such that $1 = ak + nm$ for some integer m.
    $g : \mathbb{N}_n \to \mathbb{N}_n$ is defined by $g(x) = (kx + c) \bmod n$.

    Prove that $g = f^{-1}$.

2.4 Countability

Let's have a short discussion about counting sets that may not be finite. We'll have to examine what it means to count an infinite set and what it means to compare the size of infinite sets. In so doing we'll find some useful techniques that can be applied to questions in computer science. For example, we'll see as a consequence of our discussions that there are some limits on what can be computed. We'll start with some simplifying notation.

2.4.1 Comparing the Size of Sets

Let A and B be sets. If there is a bijection between A and B, we'll denote the fact by writing

$$|A| = |B|.$$

In this case we'll say that A and B have the *same size* or have the *same cardinality*, or are *equipotent*.

example **2.19 Cardinality of a Finite Set**

Let $A = \{(x + 1)^3 \mid x \in \mathbb{N} \text{ and } 1 \leq (x + 1)^3 \leq 3000\}$. Let's find the cardinality of A. After a few calculations we can observe that

$$(0 + 1)^3 = 1,\ (1 + 1)^3 = 8,\ \ldots,\ (13 + 1)^3 = 2744 \text{ and } (14 + 1)^3 = 3375.$$

So we have a bijection $f : \{0, 1, \ldots, 13\} \to A$, where $f(x) = (x + 1)^3$. Therefore, $|A| = |\{0, 1, \ldots, 13\}| = 14$.

end example

example **2.20 Cardinality of an Infinite Set**

Let *Odd* denote the set of odd natural numbers. Then the function $f : \mathbb{N} \to Odd$ defined by $f(x) = 2x + 1$ is a bijection. So *Odd* and $\mathbb{N}$ have the same size and we write $|Odd| = |\mathbb{N}|$.

end example

If there is an injection from A to B, we'll denote the fact by writing

$$|A| \leq |B|.$$

In this case we'll say that the size, or cardinality, of A is *less than or the same as* that of B. Recall that there is an injection from A to B if and only if there is a surjection from B to A. So $|A| \leq |B|$ also means that there is a surjection from B to A.

If there is an injection from A to B but *no* bijection between them, we'll denote the fact by writing

$$|A| < |B|.$$

In this case we'll say that the size, or cardinality, of A *is less than* that of B. So $|A| < |B|$ means that $|A| \leq |B|$ and $|A| \neq |B|$.

2.4.2 Sets that Are Countable

Informally, a set is countable if its elements can be counted in a step by step fashion (e.g., count one element each second), even if it takes as many seconds as there are natural numbers. Let's clarify the idea by relating sets that can be counted to subsets of the natural numbers.

If A is a finite set with n elements, then we can represent the elements of A by listing them in the following sequence:

$$x_0, x_1, x_2, \ldots, x_{n-1}.$$

If we associate each x_k with the subscript k, we get a bijection between A and the set $\{0, 1, \ldots, n - 1\}$.

Suppose A is an infinite set such that we can represent all the elements of A by listing them in the following infinite sequence:

$$x_0, x_1, x_2, \ldots, x_n, \ldots .$$

If we associate each x_k with the subscript k, we get a bijection between A and the set $\mathbb{N}$ of natural numbers.

Definition of Countable

The preceding descriptions give us the seeds for a definition of countable. A set is *countable* if it is finite or if there is a bijection between it and $\mathbb{N}$. In the latter case, the set is said to be *countably infinite*. In terms of size we can say that a set S is countable if $|S| = |\{0, 1, \ldots, n - 1\}|$ for some natural number n or $|S| = |\mathbb{N}|$. If a set is not countable, it is said to be *uncountable.*

Countable Properties

a. Every subset of $\mathbb{N}$ is countable.

b. S is countable if and only if $|S| \leq |\mathbb{N}|$.

c. Any subset of a countable set is countable.

d. Any image of a countable set is countable.

Proof. We'll prove (a) and (b) and leave (c) and (d) as exercises. Let S be a subset of $\mathbb{N}$. If S is finite, then it is countable by definition. So assume that S is infinite. Now since S is a set of natural numbers, it has a smallest element that we'll represent by x_0. Next, we'll let x_1 be the smallest element of the set $S - \{x_0\}$. We'll continue in this manner, letting x_n be the smallest element of $S - \{x_0, \ldots, x_{n-1}\}$. In this way we obtain an infinite listing of the elements of S:

$$x_0, x_1, x_2, \ldots, x_n, \ldots.$$

Notice that each element $m \in S$ is in the listing because there are at most m elements of S that are less than m. So m must be represented as one of the elements $x_0, x_1, x_2, \ldots, x_m$ in the sequence. The association x_k to k gives a bijection between S and $\mathbb{N}$. So $|S| = |\mathbb{N}|$ and thus S is countable.

(b) If S is countable then $|S| \leq |\mathbb{N}|$ by definition. So assume that $|S| \leq |\mathbb{N}|$. Then there is an injection $f : S \rightarrow \mathbb{N}$. So $|S| = |f(S)|$. Since $f(S)$ is a subset of $\mathbb{N}$, it is countable by (a). Therefore $f(S)$ is either finite or $|f(S)|=|\mathbb{N}|$. So S is either finite or $|S| = |f(S)|=|\mathbb{N}|$. QED

Techniques to Show Countability

An interesting and useful fact about countablity is that the set $\mathbb{N} \times \mathbb{N}$ is countable. We'll state it for the record.

Theorem **(2.9)**

$\mathbb{N} \times \mathbb{N}$ is a countable set.

Proof: We need to describe a bijection between $\mathbb{N} \times \mathbb{N}$ and $\mathbb{N}$. We'll do this by arranging the elements of $\mathbb{N} \times \mathbb{N}$ in such a way that they can be easily counted. One way to do this is shown in the following listing, where each row lists a sequence of tuples in $\mathbb{N} \times \mathbb{N}$ followed by a corresponding sequence of natural numbers.

$$\begin{array}{lcl}
(0,0), & \longleftrightarrow & 0, \\
(0,1),(1,0), & \longleftrightarrow & 1,2 \\
(0,2),(1,1),(2,0), & \longleftrightarrow & 3,4,5, \\
\vdots & \vdots & \vdots \\
(0,n),\cdots & \longleftrightarrow & (n^2+n)/2,\cdots \\
\vdots & \vdots & \vdots
\end{array}$$

Notice that each row of the listing contains all the tuples whose components add up to the same number. For example, the sequence (0, 2), (1, 1), (2, 0) consists of all tuples whose components add up to 2. So we have a bijection between $\mathbb{N} \times \mathbb{N}$ and $\mathbb{N}$. Therefore, $\mathbb{N} \times \mathbb{N}$ is countable. QED.

We should note that the bijection described in (2.9) is called *Cantor's pairing function.* It maps each pair of natural numbers (x, y) to the natural number

$$\frac{(x+y)^2+3x+y}{2}.$$

We can use (2.9) to prove the following result that the union of a countable collection of countable sets is countable.

Counting Unions of Countable Sets **(2.10)**

If S_0, S_1, ..., S_n, ... is a sequence of countable sets, then the union

$$S_0 \cup S_1 \cup \cdots \cup S_n \cup \cdots$$

is a countable set.

Proof: Since each set is countable, its elements can be indexed by natural numbers. So for each set S_n we'll list its elements as x_{n0}, x_{n1}, x_{n2}, ... If S_n is a finite set then we'll list one of its elements repeatedly to make the listing infinite. In the same way, if there are only finitely many sets, then we'll list one of the sets repeatedly to make the sequence infinite. In this way we can associate each tuple (m, n) in $\mathbb{N} \times \mathbb{N}$ with an element x_{mn} in the union of the given sets. The mapping may not be a bijection since some elements of the union might be repeated in the listings. But the mapping is a surjection from $\mathbb{N} \times \mathbb{N}$ to the union of the sets. So, since $\mathbb{N} \times \mathbb{N}$ is countable, it follows that the union is countable. QED.

example **2.21 Countability of the Rationals**

We'll show that the set $\mathbb{Q}$ of rational numbers is countable by showing that $|\mathbb{Q}| = |\mathbb{N}|$. Let $\mathbb{Q}^+$ denote the set of positive rational numbers. So we can represent $\mathbb{Q}^+$ as the following set of fractions, where repetitions are included (e.g., 1/1 and 2/2 are both elements of the set).

$$\mathbb{Q}^+ = \{m/n \mid m, n \in \mathbb{N} \text{ and } n \neq 0\}.$$

Now we'll associate each fraction m/n with the tuple (m, n) in $\mathbb{N} \times \mathbb{N}$. This association is an injection, so we have $|\mathbb{Q}^+| \leq |\mathbb{N} \times \mathbb{N}|$. Since $\mathbb{N} \times \mathbb{N}$ is countable, it follows that $\mathbb{Q}^+$ is countable. In the same way, the set $\mathbb{Q}^-$ of negative rational numbers is countable. Now we can write $\mathbb{Q}$ as the union of three countable sets:

$$\mathbb{Q} = \mathbb{Q}^+ \cup \{0\} \cup \mathbb{Q}^-.$$

Since each set in the union is countable, it follows from (2.10) that the union of the sets is countable.

end example

Counting Strings

An important consequence of (2.10) is the following fact about the countability of the set of all strings over a finite alphabet.

Theorem **(2.11)**

The set A^* of all strings over a finite alphabet A is countably infinite.

Proof: For each $n \in \mathbb{N}$, let A_n be the set of strings over A having length n. It follows that A^* is the union of the sets $A_0, A_1, \ldots, A_n, \ldots$. Since each set A_n is finite, we can apply (2.10) to conclude that A^* is countable. QED.

2.4.3 Diagonalization

Let's discuss a classic construction technique, called *diagonalization*, which is quite useful in several different settings that deal with counting. The technique was introduced by Cantor when he showed that the real numbers are uncountable. Here is a general description of diagonalization.

Diagonalization **(2.12)**

Let A be an alphabet with two or more symbols and let $S_0, S_1, \ldots, S_n, \ldots,$ be a countable listing of sequences of the form $S_n = (a_{n0}, a_{n1}, \ldots, a_{nn}, \ldots)$, where $a_{ni} \in A$. The sequences are listed as the rows of the following infinite matrix.

Continued ➡

$$\begin{array}{c|cccccc}
 & 0 & 1 & 2 & \cdots & n & \cdots \\
\hline
S_0 & a_{00} & a_{01} & a_{02} & \cdots & a_{0n} & \cdots \\
S_1 & a_{10} & a_{11} & a_{12} & \cdots & a_{1n} & \cdots \\
S_2 & a_{20} & a_{21} & a_{22} & \cdots & a_{2n} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
S_n & a_{n0} & a_{n1} & a_{n2} & \cdots & a_{nn} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

Then there is a sequence $S = (a_0, a_1, a_2, \ldots, a_n, \ldots)$ over A that is *not* in the original list. We can construct S from the list of diagonal elements (a_{00}, a_{11}, a_{22}, ..., a_{nn}, ...) by changing each element so that $a_n \neq a_{nn}$ for each n. Therefore, S differs from each S_n at the nth element. For example, pick two elements x, $y \in A$ and define

$$a_n = \begin{cases} x & \text{if } a_{nn} = y \\ y & \text{if } a_{nn} \neq y. \end{cases}$$

Uncountable Sets

Now we're in position to give some examples of uncountable sets. We'll demonstrate the method of Cantor, which uses proof by contradiction together with the diagonalization technique.

example 2.22 Uncountability of the Reals

We'll show that the set $\mathbb{R}$ of real numbers is uncountable. It was shown in Exercise 6e of Section 2.3 that there is a bijection between R and the set U of real numbers between 0 and 1. So $|\mathbb{R}| = |U|$ and we need only show that U is uncountable. Assume, by way of contradiction, that U is countable. Then we can list all the numbers between 0 and 1 as a countable sequence

$$r_0,\ r_1,\ r_2,\ \ldots,\ r_n,\ \ldots.$$

Each real number in between 0 and 1 can be represented as an infinite decimal. So for each n there is a representation $r_n = 0.d_{n0}d_{n1}\ldots d_{nn}\ldots$, where each d_{ni} is a decimal digit. Since we can also represent r_n by the sequence of decimal digits $(d_{n0}d_{n1}\ldots d_{nn}\ldots)$, it follows by diagonalization (2.12) that there is an infinite decimal that is not in the list. For example, we'll choose the digits 1 and

2 to construct the number $s = 0.s_0 s_1 s_2 \ldots$ where

$$s_k = \begin{cases} 1 & \text{if } d_{kk} = 2 \\ 2 & \text{if } d_{kk} \neq 2. \end{cases}$$

So $0 < s < 1$ and s differs from each r_n at the nth decimal place. Thus s is not in the list, contrary to our assumption that we have listed all numbers in U. So the set U is uncountable, and hence also $\mathbb{R}$ is uncountable.

end example

example

2.23 Natural Number Functions

How many different functions are there from $\mathbb{N}$ to $\mathbb{N}$? We'll show that the set of all such functions is uncountable. Assume, by way of contradiction, that the set is countable. Then we can list all the functions type $\mathbb{N} \to \mathbb{N}$ as $f_0, f_1, \ldots f_n, \ldots$. We'll represent each function f_n as the sequence of its values $(f_n(0), f_n(1), \ldots, f_n(n), \ldots)$. Now (2.12) tells us there is a function missing from the list, which contradicts our assumption that all functions are in the list. So the set of all functions of type $\mathbb{N} \to \mathbb{N}$ is uncountable.

For example, we might choose the numbers $1, 2 \in \mathbb{N}$ and define a function $g : \mathbb{N} \to \mathbb{N}$ by

$$g(n) = \begin{cases} 1 & \text{if } f_n(n) = 2 \\ 2 & \text{if } f_n(n) \neq 2 \end{cases}$$

Then the sequence of values $(g(0), g(1), \ldots, g\ (n), \ldots)$ is different from each of the sequences for the listed functions because $g(n) \neq f_n(n)$ for each n. In this example there are many different ways to define $g : \mathbb{N} \to \mathbb{N}$ so that it is not in the list. For example, instead of picking 1 and 2, we could pick any two natural numbers to define g. We could also define g by

$$g(n) = f_n(n) + 1.$$

This definition gives us a function g from $\mathbb{N}$ to $\mathbb{N}$ such such that $g(n) \neq f_n(n)$ for each n. So g cannot be in the list $f_0, f_1, \ldots f_n, \ldots$.

end example

2.4.4 Limits on Computability

Let's have a short discussion about whether there are limits on what can be computed. As another application of (2.11) we can answer the question: How many programs can be written in your favorite programming language? The answer is countably infinite. Here is the result.

Theorem **(2.13)**

The set of programs for a programming language is countably infinite.

Proof: One way to see this is to consider each program as a finite string of symbols over a fixed finite alphabet A. For example, A might consist of all characters that can be typed from a keyboard. Now we can proceed as in the proof of (2.11). For each natural number n, let P_n denote the set of all programs that are strings of length n over A. For example, the program

$$\{\text{print('help')}\}$$

is in P_{15} because it's a string of length 15. So the set of all programs is the union of the sets $P_0, P_1, \ldots, P_n, \ldots$. Since each P_n is finite, hence countable, it follows from (2.10) that the union is countable. QED

Not Everything Is Computable

Since there are "only" a countable number of computer programs, it follows that there are limits on what can be computed. For example, there are an uncountable number of functions of type $\mathbb{N} \to \mathbb{N}$. So there are programs to calculate only a countable subset of these functions.

Can any real number be computed to any given number of decimal places? The answer is no. The reason is that there are "only" a countable number of computer programs (2.13) but the set of real numbers is uncountable. Therefore, there are only a countable number of computable numbers in $\mathbb{R}$ because each computable number needs a program to compute it. If we remove the computable numbers from $\mathbb{R}$, the resulting set is still uncountable. Can you see why? So most real numbers cannot be computed.

The rational numbers can be computed, and there are also many irrational numbers that can be computed. Pi is the most famous example of a computable irrational number. In fact, there are countably infinitely many computable irrational numbers.

Higher Cardinalities

It's easy to find infinite sets having many different cardinalities because Cantor proved that there are more subsets of a set than there are elements of the set. In other words, for any set A, we have the following result.

Theorem **(2.14)**

$$|A| < |\text{power}(A)|.$$

We know this is true for finite sets. But it's also true for infinite sets. We'll discuss the proof in an exercise. Notice that if A is countably infinite, then we can conclude from (2.14) that power(A) is uncountable. So, for example, we can conclude that power(N) is uncountable.

For another example, we might wonder how many different languages there are over a finite alphabet such as $\{a, b\}$. Since a language over $\{a, b\}$ is a set of strings over $\{a, b\}$, it follows that such a language is a subset of $\{a, b\}^*$, the set of all strings over $\{a, b\}$. So the set of all languages over $\{a, b\}$ is power($\{a, b\}^*$). From (2.11) we can conclude that $\{a, b\}^*$ is countably infinite. In other words, we have $|\{a, b\}^*| = |\mathbb{N}|$. So we can use (2.14) to obtain

$$|\mathbb{N}| = |\{a, b\}^*| < |\text{power}(\{a, b\}^*)|.$$

Therefore, power($\{a, b\}^*$) is uncountable, which is the same as saying that there are uncountably many languages over the alphabet $\{a, b\}$. Of course, this generalizes to any finite alphabet. So we have the following statement.

Theorem **(2.15)**

There are uncountably many languages over a finite alphabet.

We can use (2.14) to find infinite sequences of sets of higher and higher cardinality. For example, we have

$$|\mathbb{N}| < |\text{power}(\mathbb{N})| < |\text{power}(\text{power}(\mathbb{N}))| < \cdots$$

Can we associate these sets with more familiar sets? Sure, it can be shown that $|\mathbb{R}| = |\text{power}(\mathbb{N})|$, which we'll discuss in an exercise. So we have

$$|\mathbb{N}| < |\mathbb{R}| < |\text{power}(\text{power}(\mathbb{N}))| < \cdots$$

Is there any "well-known" set S such that $|S| = |\text{power}(\text{power}(\mathbb{N}))|$? Since the real numbers are hard enough to imagine, how can we comprehend all the elements in power(power($\mathbb{N}$))? Luckily, in computer science we will seldom, if ever, have occasion to worry about sets having higher cardinality than the set of real numbers.

The Continuum Hypothesis

We'll close the discussion with a question: Is there a set S whose cardinality is between that of $\mathbb{N}$ and that of the real numbers $\mathbb{R}$? In other words, does there exist a set S such that $|\mathbb{N}| < |S| < |\mathbb{R}|$? The answer is that no one knows. Interestingly, it has been shown that people who assume that the answer is yes won't run into any contradictions by using the assumption in their reasoning. Similarly, it has been shown that people who assume that the answer is no won't run into any contradictions by using the assumption in their arguments! The assumption that the answer is no is called the *continuum hypothesis.*

If we accept the continuum hypothesis, then we can use it as part of a proof technique. For example, suppose that for some set S we can show that $|\mathbb{N}| \leq |S| < |\mathbb{R}|$. Then we can conclude that $|\mathbb{N}| = |S|$ by the continuum hypothesis.

Exercises

Finite Sets

1. Find the cardinality of each set by establishing a bijection between it and a set of the form $\{0, 1, \ldots, n\}$.

   a. $\{2x + 5 \mid x \in \mathbb{N} \text{ and } 1 \leq 2x + 5 \leq 100\}$.
   b. $\{x^2 \mid x \in \mathbb{N} \text{ and } 0 \leq x^2 \leq 500\}$.
   c. $\{2, 5, 8, 11, 14, 17, \ldots, 44, 47\}$.

Countable Infinite Sets

2. Show that each of the following sets is countable by establishing a bijection between the set and $\mathbb{N}$.

   a. The set of even natural numbers.
   b. The set of negative integers.
   c. The set of strings over $\{a\}$.
   d. The set of lists over $\{a\}$ that have even length.
   e. The set $\mathbb{Z}$ of integers.
   f. The set of odd integers.
   g. The set of even integers.

3. Use (2.10) to show that each of the following sets is countable by describing the set as as a union of countable sets.

   a. The set of strings over $\{a, b\}$ of that have odd length.
   b. The set of all lists over $\{a, b\}$.
   c. The set of all binary trees over $\{a, b\}$.
   d. $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$.

Diagonalization

4. For each countable set of infinite sequences, use diagonalization (2.12) to construct an infinite sequence of the same type that is not in the set.

   a. $\{(f_n(0), f_n(1), \ldots, f_n(n), \ldots)| f_n(k) \in \{\text{hello, world}\} \text{ for } n, k \in \mathbb{N}\}$.
   b. $\{(f(n, 0), f(n, 1), \ldots, f(n, n), \ldots) \mid f(n, k) \in \{a, b, c\} \text{ for } n, k \in \mathbb{N}\}$.
   c. $\{\{a_{n0}, a_{n1}, \ldots, a_{nn}, \ldots\} \mid a_{nk} \in \{2, 4, 6, 8\} \text{ for } n, k \in \mathbb{N}\}$.

5. To show that power($\mathbb{N}$) is uncountable, we can proceed by way of contradiction. Assume that it is countable, so that all the subsets of $\mathbb{N}$ can be listed S_0, S_1, S_2, ..., S_n, ... Complete the proof by finding a way to represent each subset of $\mathbb{N}$ as an infinite sequence of 1's and 0's, where 1 means true and 0 means false. Then a contradiction arises by using diagonalization (2.12) to construct an infinite sequence of the same type that represents a subset of $\mathbb{N}$ that is not listed.

Proofs and Challenges

6. Show that if A is uncountable and B is a countable subset of A, then the set $A - B$ is uncountable.

7. Prove each statement about countable sets:
   a. Any subset of a countable set is countable.
   b. Any image of a countable set is countable.

8. Let A be a countably infinite alphabet $A = \{a_0, a_1, a_2, \ldots\}$. Let A^* denote the set of all strings over A. For each $n \in \mathbb{N}$, let A_n denote the set of all strings in A^* having length n.
   a. Show that A_n is countable for $n \in \mathbb{N}$. *Hint:* Use (2.10).
   b. Show that A^* is countable. *Hint:* Use (2.10) and part (a).

9. Let finite($\mathbb{N}$) denote the set of all finite subsets of $\mathbb{N}$. Use (2.10) to show that finite($\mathbb{N}$) is countable.

10. We'll start a proof that $|A| < |\text{power}(A)|$ for any set A. Proof: Since each element $x \in A$ can be associated with $\{x\} \in \text{power}(A)$, it follows that $|A| \leq |\text{power}(A)|$. To show that $|A| < |\text{power}(A)|$ we'll assume, by way of contradiction, that there is a bijection $A \to \text{power}(A)$. So each $x \in A$ is associated with a subset S_x of A. Now, define the following subset of A.

$$S = \{x \in A \mid x \notin S_x\}.$$

Since S is a subset of A, our assumed bijection tells us that there must be an element y in A that is associated with S. In other words, $S_y = S$. Find a contradiction by observing where y is and where it is not.

2.5 Chapter Summary

Functions allow us to associate different sets of objects. They are characterized by associating each domain element with a unique codomain element. For any function $f : A \to B$, subsets of the domain A have images in the codomain B

and subsets of B have pre-images in A. The image of A is the range of f. Partial functions need not be defined for all domain elements.

Some functions that are particularly useful in computer science are floor, ceiling, greatest common divisor (with the associated division algorithm), mod, and log.

Composition is a powerful tool for constructing new functions from known functions. Three functions that are useful in programming with lists are sequence, distribute, and pairs. The map function is a useful tool for computing lists of values of a function.

Three important properties of functions that allow us to compare sets are injective, surjective, and bijective. These properties are useful in describing the pigeonhole principle and in working with ciphers and hash functions. These properties are also useful in comparing the cardinality of sets.

A set is countable if it is finite or has the same cardinality as the set of natural numbers. Countable unions of countable sets are countable. The set of all computer programs is countable. The diagonalization technique can be used to show that some countable listings are not exhaustive. It can also be used to show that some sets, such as the real numbers, are uncountable. So we can't compute all the real numbers. Any set has smaller cardinality than its power set, even when the set is infinite.

chapter 3

Construction Techniques

When we build, let us think that we build forever.

—John Ruskin (1819–1900)

To construct an object, we need some kind of description. If we're lucky, the description might include a construction technique. Otherwise, we may need to use our wits and our experience to construct the object. This chapter focuses on gaining some construction experience.

The only way to learn a technique is to use it on a wide variety of problems. We'll present each technique in the framework of objects that occur in computer science, and as we go along, we'll extend our knowledge of these objects. We'll begin by introducing the technique of inductive definition for sets. Then we'll discuss techniques for describing recursively defined functions and procedures. Last but not least, we'll introduce grammars for describing sets of strings.

There are usually two parts to solving a problem. The first is to guess at a solution and the second is to verify that the guess is correct. The focus of this chapter is to introduce techniques to help us make good guesses. We'll usually check a few cases to satisfy ourselves that our guesses are correct. In the next chapter we'll study inductive proof techniques that can be used to actually prove correctness of claims about objects constructed by the techniques of this chapter.

chapter guide

Section 3.1 introduces the inductive definition technique. We'll apply the technique by defining various sets of numbers, strings, lists, binary trees, and Cartesian products.

Section 3.2 introduces the technique of recursive definition for functions and procedures. We'll apply the technique to functions and procedures that process numbers, strings, lists, and binary trees. We'll solve the repeated element problem and the power set problem, and we'll construct some functions for infinite sequences.

Section 3.3 introduces the idea of a grammar as a way to describe a language. We'll see that grammars describe the strings of a language in an inductive fashion, and we'll see that they provide recursive rules for testing whether a string belongs to a language.

3.1 Inductively Defined Sets

When we write down an informal statement such as $A = \{3, 5, 7, 9, \ldots\}$, most of us will agree that we mean the set $A = \{2k + 3 \mid k \in \mathbb{N}\}$. Another way to describe A is to observe that $3 \in A$, that $x \in A$ implies $x + 2 \in A$, and that the only way an element gets in A is by the previous two steps. This description has three ingredients, which we'll state informally as follows:

1. There is a starting element (3 in this case).
2. There is a construction operation to build new elements from existing elements (addition by 2 in this case).
3. There is a statement that no other elements are in the set.

Definition of Inductive Definition

This process is an example of an *inductive definition* of a set. The set of objects defined is called an *inductive set.* An inductive set consists of objects that are constructed, in some way, from objects that are already in the set. So nothing can be constructed unless there is at least one object in the set to start the process. Inductive sets are important in computer science because the objects can be used to represent information and the construction rules can often be programmed. We give the following formal definition.

An inductive definition of a set S consists of three steps: **(3.1)**

Basis: Specify one or more elements of S.

Induction: Give one or more rules to construct new elements of S from existing elements of S.

Closure: State that S consists exactly of the elements obtained by the basis and induction steps. This step is usually assumed rather than stated explicitly.

The closure step is a very important part of the definition. Without it, there could be lots of sets satisfying the first two steps of an inductive definition. For example, the two sets $\mathbb{N}$ and $\{3, 5, 7, \ldots\}$ both contain the number 3, and if x

is in either set, then so is $x + 2$. It's the closure statement that tells us that the only set defined by the basis and induction steps is $\{3, 5, 7, \dots\}$. So the closure statement tells us that we're defining exactly one set, namely, the smallest set satisfying the basis and induction steps. We'll always omit the specific mention of closure in our inductive definitions.

The *constructors* of an inductive set are the basis elements and the rules for constructing new elements. For example, the inductive set $\{3, 5, 7, 9, \dots\}$ has two constructors, the number 3 and the operation of adding 2 to a number.

For the rest of this section we'll use the technique of inductive definition to construct sets of objects that are often used in computer science.

3.1.1 Numbers

The set of natural numbers $\mathbb{N} = \{0, 1, 2, \dots\}$ is an inductive set. Its basis element is 0, and we can construct a new element from an existing one by adding the number 1. So we can write an inductive definition for $\mathbb{N}$ in the following way.

Basis: $0 \in \mathbb{N}$.

Induction: If $n \in \mathbb{N}$, then $n + 1 \in \mathbb{N}$.

The constructors of $\mathbb{N}$ are the integer 0 and the operation that adds 1 to an element of $\mathbb{N}$. The operation of adding 1 to n is called the *successor* function, which we write as

$$\text{succ}(n) = n + 1.$$

Using the successor function, we can rewrite the induction step in the above definition of $\mathbb{N}$ in the alternative form

$$\text{If } n \in \mathbb{N}, \text{ then succ}(n) \in \mathbb{N}.$$

So we can say that $\mathbb{N}$ is an inductive set with two constructors, 0 and succ.

example 3.1 Some Familiar Odd Numbers

We'll give an inductive definition of $A = \{1, 3, 7, 15, 31, \dots\}$. Of course, the basis case should place 1 in A. If $x \in A$, then we can construct another element of A with the expression $2x + 1$. So the constructors of A are the number 1 and the operation of multiplying by 2 and adding 1. An inductive definition of A can be written as follows:

Basis: $1 \in A$.

Induction: If $x \in A$, then $2x + 1 \in A$.

end example

example

3.2 Some Even and Odd Numbers

Is the following set inductive?

$$A = \{2, 3, 4, 7, 8, 11, 15, 16, \ldots\}.$$

It might be easier if we think of A as the union of the two sets

$$B = \{2, 4, 8, 16, \ldots\} \text{ and } C = \{3, 7, 11, 15, \ldots\}.$$

Both these sets are inductive. The constructors of B are the number 2 and the operation of multiplying by 2. The constructors of C are the number 3 and the operation of adding by 4. We can combine these definitions to give an inductive definition of A.

Basis: $2, 3 \in A$.

Induction: If $x \in A$ and x is odd, then $x + 4 \in A$.
If $x \in A$ and x is even, then $2x \in A$.

This example shows that there can be more than one basis element, more than one induction rule, and tests can be included.

end example

example

3.3 Communicating with a Robot

Suppose we want to communicate the idea of the natural numbers to a robot that knows about functions, has a loose notion of sets, and can follow an inductive definition. Symbols like 0, 1, ..., and + make no sense to the robot. How can we convey the idea of $\mathbb{N}$? We'll tell the robot that N is the name of the set we want to construct.

Suppose we start by telling the robot to put the symbol 0 in N. For the induction case we need to tell the robot about the successor function. We tell the robot that $s : N \to N$ is a function, and whenever an element $x \in N$, then put the element $s(x) \in N$. After a pause, the robot says, "$N = \{0\}$ because I'm letting s be the function defined by $s(0) = 0$."

Since we don't want $s(0) = 0$, we have to tell the robot that $s(0) \neq 0$. Then the robot says, "$N = \{0, s(0)\}$ because $s(s(0)) = 0$." So we tell the robot that $s(s(0)) \neq 0$. Since this could go on forever, let's tell the robot that $s(x)$ does not equal any previously defined element. Do we have it? Yes. The robot responds with "$N = \{0, s(0), s(s(0)), s(s(s(0))), \ldots\}$." So we can give the robot the following definition:

Basis: $0 \in N$.

Induction: If $x \in N$, then put $s(x) \in N$, where $s(x) \neq 0$ and $s(x)$ is not equal to any previously defined element of N.

This definition of the natural numbers—along with a closure statement—is due to the mathematician and logician Giuseppe Peano (1858–1932).

end example

example 3.4 Communicating with Another Robot

Suppose we want to define the natural numbers for a robot that knows about sets and can follow an inductive definition. How can we convey the idea of $\mathbb{N}$ to the robot? Since we can use only the notation of sets, let's use $\varnothing$ to stand for the number 0.

What about the number 1? Can we somehow convey the idea of 1 using the empty set? Let's let $\{\varnothing\}$ stand for 1. What about 2? We can't use $\{\varnothing, \varnothing\}$, because $\{\varnothing, \varnothing\} = \{\varnothing\}$. Let's let $\{\varnothing, \{\varnothing\}\}$ stand for 2 because it has two distinct elements. Notice the little pattern we have going: If s is the set standing for a number, then $s \cup \{s\}$ stands for the successor of the number.

Starting with $\varnothing$ as the basis element, we have an inductive definition. Letting *Nat* be the set that we are defining for the robot, we have the following inductive definition.

Basis: $\varnothing \in$ *Nat.*

Induction: If $s \in$ *Nat*, then $s \cup \{s\} \in$ *Nat.*

For example, since 2 is represented by the set $\{\varnothing, \{\varnothing\}\}$, the number 3 is represented by the set

$$\{\varnothing, \{\varnothing\}\} \cup \{\{\varnothing, \{\varnothing\}\}\} = \{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}\}.$$

This is not fun. After a while we might try to introduce some of our own notation to the robot. For example, we'll introduce the decimal numerals in the following way.

$$\begin{aligned} 0 &= \varnothing, \\ 1 &= 0 \cup \{0\}, \\ 2 &= 1 \cup \{1\}, \\ &\vdots \end{aligned}$$

Now we can think about the natural numbers in the following way.

$$\begin{aligned} 0 &= \varnothing, \\ 1 &= 0 \cup \{0\} \varnothing \cup \{0\} = \{0\}, \\ 2 &= 1 \cup \{1\} = \{0\} \cup \{1\} = \{0, 1\}, \\ &\vdots \end{aligned}$$

Therefore, each number is the set of numbers that precede it.

end example

3.1.2 Strings

We often define strings of things inductively without even thinking about it. For example, in high school algebra we might say that an algebraic expression is either a number or a variable, and if A and B are algebraic expressions, then so are (A), $A + B$, $A - B$, AB, and $A \div B$. So the set of algebraic expressions is a set of strings. For example, if x and y are variables, then the following strings are algebraic expressions.

$$x, \quad y, \quad 25, \quad 25x, \quad x + y, \quad (4x + 5y), \quad (x + y)(2yx), \quad 3x \div 4.$$

If we like, we can make our definition more formal by specifying the basis and induction parts. For example, if we let E denote the set of algebraic expressions as we have described them, then we have the following inductive definition for E.

Basis: If x is a variable or a number, then $x \in E$.

Induction: If $A, B \in E$, then (A), $A + B$, $A - B$, AB, $A \div B \in E$.

Let's recall that for an alphabet A, the set of all strings over A is denoted by A^*. This set has the following inductive definition.

All Strings over A **(3.2)**

Basis: $\Lambda \in A^*$.

Induction: If $s \in A^*$ and $a \in A$, then $as \in A^*$.

We should note that when we place two strings next to each other in juxtaposition to form a new string, we are concatenating the two strings. So, from a computational point of view, concatenation is the operation we are using to construct new strings.

Recall that any set of strings is called a *language*. If A is an alphabet, then any language over A is one of the subsets of A^*. Many languages can be defined inductively. Here are some examples.

example 3.5 Three Languages

We'll give an inductive definition for each of three languages.

1. $S = \{a, ab, abb, abbb, \ldots\} = \{ab^n \mid n \in \mathbb{N}\}$.

Informally, we can say that the strings of S consist of the letter a followed by zero or more b's. But we can also say that the letter a is in S, and if x is a string in S, then so is xb. This gives us an inductive definition for S.

Basis: $a \in S$.

Induction: If $x \in S$, then $xb \in S$.

2. $S = \{\Lambda \text{ , } ab, aabb, aaabbb, \ldots\} = \{a^n b^n \mid n \in \mathbb{N}\}$.

Informally, we can say that the strings of S consist of any number of a's followed by the same number of b's. But we can also say that the empty string Λ is in S, and if x is a string in S, then so is axb. This gives us an inductive definition for S.

Basis: $\Lambda \in S$.

Induction: If $x \in S$, then $axb \in S$.

3. $S = \{\Lambda \text{ , } ab, abab, ababab, \ldots\} = \{(ab)^n \mid n \in \mathbb{N}\}$.

Informally, we can say that the strings of S consist of any number of ab pairs. But we can also say that the empty string Λ is in S, and if x is a string in S, then so is abx. This gives us an inductive definition for S.

Basis: $\Lambda \in S$.

Induction: If $x \in S$, then $abx \in S$.

end example

example

3.6 Decimal Numerals

Let's give an inductive definition for the set of decimal numerals. Recall that a decimal numeral is a nonempty string of decimal digits. For example, 2340 and 002965 are decimal numerals. If we let D denote the set of decimal numerals, we can describe D by saying that any decimal digit is in D, and if x is in D and d is a decimal digit, then dx is in D. This gives us the following inductive definition for D:

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

Induction: If $x \in D$ and d is a decimal digit, then $dx \in D$.

end example

3.1.3 Lists

Recall that a list is an ordered sequence of elements. Let's try to find an inductive definition for the set of lists with elements from a set A. In Chapter 1 we denoted the set of all lists over A by lists(A), and we'll continue to do so. We also mentioned that from a computational point of view the only parts of a nonempty list that can be accessed randomly are its *head* and its *tail*. Head and tail are sometimes called *destructors*, since they are used to destroy a list (take it apart). For example, the list $\langle x, y, z\rangle$ has x as its head and $\langle y, z\rangle$ as its tail, which we write as

$$\text{head}(\langle x, y, z\rangle) = x \quad \text{and} \quad \text{tail}(\langle x, y, z\rangle) = \langle y, z\rangle .$$

We also introduced the operation "cons" to construct lists, where if h is an element and t is a list, the new list whose head is h and whose tail is t is represented by the expression

$$\text{cons}(h, t).$$

So cons is a constructor of lists. For example, we have

$$\text{cons}(x, \langle y, z\rangle) = \langle x, y, z\rangle$$
$$\text{cons}(x, \langle\ \rangle) = \langle x\rangle.$$

The operations cons, head, and tail work nicely together. For example, we can write

$$\langle x, y, z\rangle = \text{cons}(x, \langle y, z\rangle) = \text{cons}(\text{head}(\langle x, y, z\rangle), \text{tail}(\langle x, y, z\rangle)).$$

So if L is any nonempty list, then we have the equation

$$L = \text{cons}(\text{head}(L), \text{tail}(L)).$$

Now we have the proper tools, so let's get down to business and write an inductive definition for lists(A). Informally, we can say that lists(A) is the set of all ordered sequences of elements taken from the set A. But we can also say that $\langle\ \rangle$ is in lists(A), and if L is in lists(A), then so is cons(a, L) for any a in A. This gives us an inductive definition for lists(A), which we can state formally as follows.

All Lists over A **(3.3)**

Basis: $\langle\ \rangle \in \text{lists}(A)$.

Induction: If $x \in A$ and $L \in \text{lists}(A)$, then $\text{cons}(x, L) \in \text{lists}(A)$.

example

3.7 List Membership

Let $A = \{a, b\}$. We'll use (3.3) to show how some lists become members of lists(A). The basis case puts $\langle\ \rangle \in$ lists(A). Since $a \in A$ and $\langle\ \rangle \in$ lists(A), the induction step gives

$$\langle a \rangle = \text{cons}(a, \langle\ \rangle) \in \text{lists}(A).$$

In the same way we get $\langle b \rangle \in$ lists(A). Now since $a \in A$ and $\langle a \rangle \in$ lists(A), the induction step puts $\langle a, a \rangle \in$ lists(A). Similarly, we get $\langle b, a \rangle$, $\langle a, b \rangle$, and $\langle b, b \rangle$ as elements of lists(A), and so on.

end example

A Notational Convenience

It's convenient when working with lists to use an infix notation for cons to simplify the notation for list expressions. We'll use the double colon symbol ::, so that the infix form of cons(x, L) is $x :: L$.

$$x :: L.$$

For example, the list $\langle a, b, c \rangle$ can be constructed using cons as

$$\begin{aligned}\text{cons}(a, \text{cons}(b, \text{cons}(c, \langle\ \rangle))) &= \text{cons}(a, \text{cons}(b, \langle c \rangle)) \\ &= \text{cons}(a, \langle b, c \rangle) \\ &= \langle a, b, c \rangle.\end{aligned}$$

Using the infix form, we construct $\langle a, b, c \rangle$ as follows:

$$a :: (b :: (c :: \langle\ \rangle)) = a :: (b :: \langle c \rangle) = a :: \langle b, c \rangle = \langle a, b, c \rangle.$$

The infix form of cons allows us to omit parentheses by agreeing that :: is right associative. In other words, $a :: b :: L = a :: (b :: L)$. Thus we can represent the list $\langle a, b, c \rangle$ by writing

$$a :: b :: c :: \langle\ \rangle \text{ instead of } a :: (b :: (c :: \langle\ \rangle)).$$

Many programming problems involve processing data represented by lists. The operations cons, head, and tail provide basic tools for writing programs to create and manipulate lists. So they are necessary for programmers. Now let's look at a few examples.

example

3.8 Lists of Binary Digits

Suppose we need to define the set S of all nonempty lists over the set $\{0, 1\}$ with the property that adjacent elements in each list are distinct. We can get an idea about S by listing a few elements:

$$S = \{\langle 0 \rangle, \langle 1 \rangle, \langle 1, 0 \rangle, \langle 0, 1 \rangle, \langle 0, 1, 0 \rangle, \langle 1, 0, 1 \rangle, \ldots\}.$$

Let's try $\langle 0 \rangle$ and $\langle 1 \rangle$ as basis elements of S. Then we can construct a new list from a list $L \in S$ by testing whether head(L) is 0 or 1. If head(L) = 0, then we place 1 at the left of L. Otherwise, we place 0 at the left of L. So we can write the following inductive definition for S.

Basis: $\langle 0 \rangle, \langle 1 \rangle \in S$.

Induction: If $L \in S$ and head$(L) = 0$, then cons$(1, L) \in S$.
If $L \in S$ and head$(L) = 1$, then cons$(0, L) \in S$.

The infix form of this induction rules looks like

If $L \in S$ and head$(L) = 0$, then $1 :: L \in S$.
If $L \in S$ and head$(L) = 1$, then $0 :: L \in S$.

end example

example

3.9 Lists of Letters

Suppose we need to define the set S of all lists over $\{a, b\}$ that begin with the single letter a followed by zero or more occurrences of b. We can describe S informally by writing a few of its elements:

$$S = \{\langle a \rangle, \langle a, b \rangle, \langle a, b, b \rangle, \langle a, b, b, b \rangle, \ldots\}.$$

It seems appropriate to make $\langle a \rangle$ the basis element of S. Then we can construct a new list from any list $L \in S$ by attaching the letter b on the right end of L. But cons places new elements at the left end of a list. We can overcome the problem in the following way:

$$\text{If } x \in S, \text{ then cons}(a, \text{cons}(b, \text{tail}(L))) \in S.$$

In infix form the statement reads as follows:

$$\text{If } x \in S, \text{ then } a :: b :: \text{tail}(L) \in S.$$

For example, if $L = \langle a \rangle$, then we construct the list

$$a :: b :: \text{tail}(\langle a \rangle) = a :: b :: \langle\, \rangle = a :: \langle b \rangle = \langle a, b \rangle.$$

So we have the following inductive definition of S:

Basis: $\langle a \rangle \in S$.

Induction: If $L \in S$, then $a :: b :: \text{tail}(L) \in S$.

end example

example 3.10 All Possible Lists

Can we find an inductive definition for the set of all possible lists over $\{a, b\}$, including lists that can contain other lists? Suppose we start with lists having a small number of symbols, including the symbols $\langle$ and $\rangle$. Then, for each $n \geq 2$, we can write down the lists made up of n symbols (not including commas). Figure 3.1 shows these listings for the first few values of n.

If we start with the empty list $\langle\ \rangle$, then with a and b we can construct three more lists as follows:

$$\begin{aligned} a :: \langle\ \rangle &= \langle a \rangle, \\ b :: \langle\ \rangle &= \langle b \rangle, \\ \langle\ \rangle :: \langle\ \rangle &= \langle\langle\ \rangle\rangle. \end{aligned}$$

Now if we take these three lists together with $\langle\ \rangle$, then with a and b we can construct many more lists. For example,

$$\begin{aligned} a :: \langle a \rangle &= \langle a, a \rangle, \\ \langle a \rangle :: \langle\ \rangle &= \langle\langle a \rangle\rangle, \\ \langle\langle\ \rangle\rangle :: \langle b \rangle &= \langle\langle\langle\ \rangle\rangle, b \rangle, \\ \langle b \rangle :: \langle\langle\ \rangle\rangle &= \langle b, \langle\ \rangle\rangle. \end{aligned}$$

Using this idea, we'll make an inductive definition for the set S of all possible lists over A.

Basis: $\langle\ \rangle \in S$. (3.4)

Induction: If $x \in A \cup S$ and $L \in S$, then $x :: L \in S$.

end example

2	3	4	5	6
$\langle\ \rangle$	$\langle a \rangle$	$\langle\langle\ \rangle\rangle$	$\langle\langle a \rangle\rangle$	$\langle\langle\langle\ \rangle\rangle\rangle$
	$\langle b \rangle$	$\langle a, a \rangle$	$\langle\langle b \rangle\rangle$	$\langle\langle\ \rangle, \langle\ \rangle\rangle$
		$\langle a, b \rangle$	$\langle\langle\ \rangle, a \rangle$	$\langle a, a, \langle\ \rangle\rangle$
		$\langle b, a \rangle$	$\langle\langle\ \rangle, b \rangle$	$\langle a, \langle\ \rangle, a \rangle$
		$\langle b, b \rangle$	$\langle a, \langle\ \rangle\rangle$	$\langle\langle\ \rangle, a, a \rangle$
			$\langle b, \langle\ \rangle\rangle$	$\langle a, b, \langle\ \rangle\rangle$
			$\langle a, a, a \rangle$	$\langle a, b, a, b \rangle$
			$\vdots$	$\vdots$

Figure 3.1 A listing of lists by size.

3.1.4 Binary Trees

Recall that a binary tree is either empty or it has a left and right subtree, each of which is a binary tree. This is an informal inductive description of the set of binary trees. To give a formal definition and to work with binary trees, we need some operations to pick off parts of a tree and to construct new trees.

In Chapter 1 we represented binary trees by lists, where the empty binary tree is denoted by $\langle\ \rangle$ and a nonempty binary tree is denoted by the list $\langle L, x, R\rangle$, where x is the root, L is the left subtree, and R is the right subtree. This gives us the ingredients for a more formal inductive definition of the set of all binary trees.

For convenience we'll let tree(L, x, R) denote the binary tree with root x, left subtree L, and right subtree R. If we still want to represent binary trees as tuples, then of course we can write

$$\text{tree}(L, x, R) = \langle L, x, R\rangle.$$

Now suppose A is any set. Then we can describe the set B of all binary trees whose nodes come from A by saying that $\langle\ \rangle$ is in B, and if L and R are in B, then so is tree(L, a, R) for any a in A. This gives us an inductive definition, which we can state formally as follows.

All Binary Trees over A **(3.5)**

Basis: $\langle\ \rangle \in B$.

Induction: If $x \in A$ and $L, R \in B$, then tree$(L, x, R) \in B$.

We also have destructor operations for binary trees. We'll let *left*, *root*, and *right* denote the operations that return the left subtree, the root, and the right subtree, respectively, of a nonempty tree. For example, if

$$T = \text{tree}(L, x, R), \text{ then left}(T) = L, \text{root}(T) = x, \text{ and right}(T) = R.$$

So for any nonempty binary tree T we have

$$T = \text{tree}(\text{left}(T), \text{root}(T), \text{right}(T)).$$

example **3.11 Binary Trees of Twins**

Let $A = \{0, 1\}$. Suppose we need to work with the set *Twins* of all binary trees T over A that have the following property: The left and right subtrees of each node in T are identical in structure and node content. For example, *Twins* contains the empty tree and any single-node tree. *Twins* also contains the two trees shown in Figure 3.2.

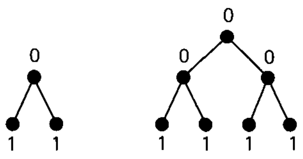


Figure 3.2 Twins as subtrees.

We can give an inductive definition of *Twins* by simply making sure that each new tree has the same left and right subtrees. Here's the definition:

Basis: $\langle\ \rangle \in$ *Twins*.

Induction: If $x \in A$ and $T \in$ *Twins*, then tree(T, x, T) $\in$ *Twins*.

end example

example **3.12 Binary Trees of Opposites**

Let $A = \{0, 1\}$, and suppose that *Opps* is the set of all nonempty binary trees T over A with the following property: The left and right subtrees of each node of T have identical structures, but the 0's and 1's are interchanged. For example, the single node trees are in *Opps*, as well as the two trees shown in Figure 3.3.

Since our set does not include the empty tree, the two singleton trees with nodes 1 and 0 should be the basis trees in *Opps*. The inductive definition of *Opps* can be given as follows:

Basis: tree($\langle\ \rangle$, 0, $\langle\ \rangle$), tree($\langle\ \rangle$, 1, $\langle\ \rangle$) $\in$ *Opps*.

Induction: Let $x \in A$ and $T \in$ *Opps*.
If root(T) = 0, then
tree(T, x, tree(right(T), 1, left(T))) $\in$ *Opps*.
Otherwise,
tree(T, x, tree(right(T), 0, left(T))) $\in$ *Opps*.

Does this definition work? Try out some examples. See whether the definition builds the four possible three-node trees.

end example

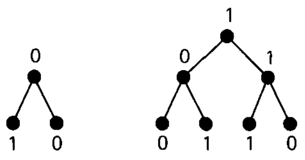


Figure 3.3 Opposites as subtrees.

3.1.5 Cartesian Products of Sets

Let's consider the problem of finding inductive definitions for subsets of the Cartesian product of two sets. For example, the set $\mathbb{N} \times \mathbb{N}$ can be defined inductively by starting with the pair (0, 0) as the basis element. Then, for any pair (x, y) in the set, we can construct the following three pairs.

$$(x + 1, y + 1), (x, y + 1), \text{ and } (x + 1, y).$$

The graph in Figure 3.4 shows an arbitrary point (x, y) together with the three new points. It seems clear that this definition will define all elements of $\mathbb{N} \times \mathbb{N}$, although some points will be defined more than once. For example, the point (1, 1) is constructed from the basis element (0, 0), but it is also constructed from the point (0, 1) and from the point (1, 0).

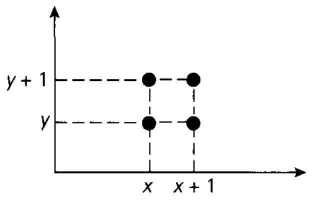


Figure 3.4 Four integer points.

example **3.13 Cartesian Product**

A Cartesian product can be defined inductively if at least one of the sets in that product can be defined inductively. For example, if A is any set, then we have the following inductive definition of $\mathbb{N} \times A$:

Basis: $(0, a) \in \mathbb{N} \times A$ for all $a \in A$.

Induction: If $(x, y) \in \mathbb{N} \times A$, then $(x + 1, y) \in \mathbb{N} \times A$.

end example

example **3.14 Part of a Plane**

Let $S = \{(x, y) | \ x, y \in \mathbb{N} \text{ and } x \leq y\}$. From the point of view of a plane, S is the set of points in the first quadrant with integer coordinates on or above the main diagonal. We can define S inductively as follows:

Basis: $(0, 0) \in S$.

Induction: If $(x, y) \in S$, then $(x, y + 1), (x + 1, y + 1) \in S$.

For example, we can use (0, 0) to construct (0, 1) and (1, 1). From (0, 1) we construct (0, 2) and (1, 2). From (1, 1) we construct (1, 2) and (2, 2). So some pairs get defined more than once.

end example

example 3.15 Describing an Area

Suppose we need to describe some area as a set of points. From a computational point of view, the area will be represented by discrete points, like pixels on a computer screen. So we can think of the area as a set of ordered pairs (x, y) forming a subset of $\mathbb{N} \times \mathbb{N}$.

To keep things simple we'll describe the the area A under the curve of a function f between two points a and b on the x-axis. Figure 3.5 shows a general picture of the area A.

So the area A can be described as the following set of points.

$$A = \{(x, y) \mid x, y \in \mathbb{N},\ a \le x \le b, \text{ and } 0 \le y \le f(x)\}.$$

There are several ways we might proceed to give an inductive definition of A. For example, we can start with the point $(a, 0)$ on the x-axis. From $(a, 0)$ we can construct the column of points above it and the point $(a + 1, 0)$, from which the next column of points can be constructed. Here's the definition.

Basis: $(a, 0) \in A$.

Induction: If $(x, 0) \in A$ and $x < b$, then $(x + 1, 0) \in A$.
If $(x, y) \in A$ and $y < f(x)$, then $(x, y + 1) \in A$.

For example, the column of points $(a, 0), (a, 1), (a, 2), \ldots, (a, f(a))$ is constructed by starting with the basis point $(a, 0)$ and by repeatedly using the second if-then statement. The first if-then statement constructs the points on the x-axis that are then used to construct the other columns of points. Notice with this definition that each pair is constructed exactly once.

end example

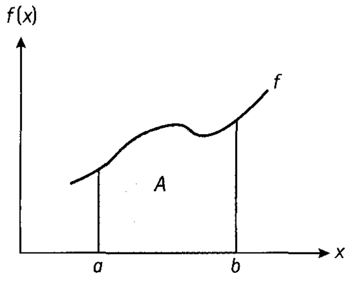


Figure 3.5 Area under a curve.

Exercises

Numbers

1. For each of the following inductive definitions, start with the basis element and construct ten elements in the set.

   a. *Basis:* $3 \in S$.
   Induction: If $x \in S$, then $2x - 1 \in S$.

   b. *Basis:* $1 \in S$.
   Induction: If $x \in S$, then $2x, 2x + 1 \in S$.

2. Find an inductive definition for each set S.

   a. $\{1, 3, 5, 7, \ldots\}$.
   b. $\{0, 2, 4, 6, 8, \ldots\}$.
   c. $\{-3, -1, 1, 3, 5, \ldots\}$.
   d. $\{\ldots, -7, -4, -1, 2, 5, 8, \ldots\}$.
   e. $\{1, 4, 9, 16, 25, \ldots\}$.
   f. $\{1, 3, 7, 15, 31, 63, \ldots\}$.

3. Find an inductive definition for each set S.

   a. $\{4, 7, 10, 13, \ldots\} \cup \{3, 6, 9, 12, \ldots\}$.
   b. $\{3, 4, 5, 8, 9, 12, 16, 17, \ldots\}$. *Hint:* Write the set as a union.

4. Find an inductive definition for each set S.

   a. $\{x \in \mathbb{N} \mid \text{floor}(x/2) \text{ is even}\}$.
   b. $\{x \in \mathbb{N} \mid \text{floor}(x/2) \text{ is odd}\}$.
   c. $\{x \in \mathbb{N} \mid x \bmod 5 = 2\}$.
   d. $\{x \in \mathbb{N} \mid 2x \bmod 7 = 3\}$.

5. The following inductive definition was given in Example 3.4, the second robot example.

   Basis: $\varnothing \in Nat$.
   Induction: If $s \in Nat$, then $s \cup \{s\} \in Nat$.

   In Example 3.4 we identified natural numbers with the elements of *Nat* by setting $0 = \varnothing$ and $n = n \cup \{n\}$ for $n \neq 0$. Show that $4 = \{0, 1, 2, 3\}$.

Strings

6. Find an inductive definition for each set S of strings.

a. $\{a^n bc^n \mid n \in \mathbb{N}\}$.
b. $\{a^{2n} \mid n \in \mathbb{N}\}$.
c. $\{a^{2n+1} \mid n \in \mathbb{N}\}$.
d. $\{a^m b^n \mid m, n \in \mathbb{N}\}$.
e. $\{a^m bc^n \mid m, n \in \mathbb{N}\}$.
f. $\{a^m b^n \mid m, n \in \mathbb{N}$, where $m > 0\}$.
g. $\{a^m b^n \mid m, n \in \mathbb{N}$, where $n > 0\}$.
h. $\{a^m b^n \mid m, n \in \mathbb{N}$, where $m > 0$ and $n > 0\}$.
i. $\{a^m b^n \mid m, n \in \mathbb{N}$, where $m > 0$ or $n > 0\}$.
j. $\{a^{2n} \mid n \in \mathbb{N}\} \cup \{b^{2n+1} \mid n \in \mathbb{N}\}$.
k. $\{s \in \{a, b\}^* \mid s$ has the same number of a's and b's$\}$.

7. Find an inductive definition for each set S of strings.

a. Even palindromes over the set $\{a, b\}$.
b. Odd palindromes over the set $\{a, b\}$.
c. All palindromes over the set $\{a, b\}$.
d. The binary numerals.

8. Let the letters a, b, and c be constants; let the letters x, y, and z be variables; and let the letters f and g be functions of arity 1. We can define the set of terms over these symbols by saying that any constant or variable is a term and if t is a term, then so are $f(t)$ and $g(t)$. Find an inductive definition for the set T of terms.

Lists

9. For each of the following inductive definitions, start with the basis element and construct five elements in the set.

a. *Basis:* $\langle a \rangle \in S$.
Induction: If $x \in S$, then $b :: x \in S$.

b. *Basis:* $\langle 1 \rangle \in S$.
Induction: If $x \in S$, then $2 \cdot \text{head}(x) :: x \in S$.

10. Find an inductive definition for each set S of lists. Use the cons constructor.

a. $\{\langle a \rangle, \langle a, a \rangle, \langle a, a, a \rangle, \ldots\}$.
b. $\{\langle 1 \rangle, \langle 2, 1 \rangle, \langle 3, 2, 1 \rangle, \ldots\}$.
c. $\{\langle a, b \rangle, \langle b, a \rangle, \langle a, a, b \rangle, \langle b, b, a \rangle, \langle a, a, a, b \rangle, \langle b, b, b, a \rangle, \ldots\}$.
d. $\{L \mid L$ has even length over $\{a\}\}$.
e. $\{L \mid L$ has even length over $\{0, 1, 2\}\}$.
f. $\{L \mid L$ has even length over a set $A\}$.

g. $\{L \mid L$ has odd length over $\{a\}\}$.
h. $\{L \mid L$ has odd length over $\{0, 1, 2\}\}$.
i. $\{L \mid L$ has odd length over a set $A\}$.

11. Find an inductive definition for each set S of lists. You may use the "consR" operation, where consR(L, x) is the list constructed from the list L by adding a new element x on the right end. Similarly, you may use the "headR" and "tailR" operations, which are like head and tail but look at things from the right side of a list.

    a. $\{\langle a\rangle, \langle a, b\rangle, \langle a, b, b\rangle, \dots\}$.
    b. $\{\langle 1\rangle, \langle 1, 2\rangle, \langle 1, 2, 3\rangle, \dots\}$.
    c. $\{ L \in \text{lists}(\{a, b\}) \mid L$ has the same number of a's and b's$\}$.

12. Find an inductive definition for the set S of all lists over $A = \{a, b\}$ that alternate a's and b's. For example, the lists $\langle\ \rangle$, $\langle a\rangle$, $\langle b\rangle$, $\langle a, b, a\rangle$, and $\langle b, a\rangle$ are in S. But $\langle a, a\rangle$ is not in S.

Binary Trees

13. Given the following inductive definition for a set S of binary trees. Start with the basis element and draw pictures of four binary trees in the set. Don't draw the empty subtrees.

    Basis: tree$(\langle\ \rangle, a, \langle\ \rangle) \in S$.
    Induction: If $T \in S$, then tree(tree$(\langle\ \rangle, a, \langle\ \rangle), a, T) \in S$.

14. Find an inductive definition for the set B of binary trees that represent arithmetic expressions that are either numbers in $\mathbb{N}$ or expressions that use operations $+$ or $-$.

15. Find an inductive definition for the set B of nonempty binary trees over $\{a\}$ in which each non-leaf node has two subtrees, one of which is a leaf and the other of which is either a leaf or a member of B.

Cartesian Products

16. Given the following inductive definition for a subset B of $\mathbb{N} \times \mathbb{N}$.

    Basis: $(0, 0) \in B$.
    Induction: If $(x, y) \in B$, then $(x + 1, y), (x + 1, y + 1) \in B$.

    a. Describe the set B as a set of the form $\{(x, y) \mid$ some property holds$\}$.
    b. Describe those elements in B that get defined in more than one way.

17. Find an inductive definition for each subset S of $\mathbb{N} \times \mathbb{N}$.

    a. $S = \{(x, y) \mid y = x$ or $y = x + 1\}$.
    b. $S = \{(x, y) \mid x$ is even and $y \leq x/2\}$.

18. Find an inductive definition for each product set S.
    a. $S = \text{lists}(A) \times \text{lists}(A)$ for some set A.
    b. $S = A \times \text{lists}(A)$.
    c. $S = \mathbb{N} \times \text{lists}(\mathbb{N})$.
    d. $S = \mathbb{N} \times \mathbb{N} \times \mathbb{N}$

Proofs and Challenges

19. Let A be a set. Suppose O is the set of binary trees over A that contain an odd number of nodes. Similarly, let E be the set of binary trees over A that contain an even number of nodes. Find inductive definitions for O and E. *Hint:* You can use O when defining E, and you can use E when defining O.

20. Use Example 3.15 as a guide to construct an inductive definition for the set of points in $\mathbb{N} \times \mathbb{N}$ that describe the area A between two curves f and g defined as follows for two natural numbers a and b:

$$A = \{(x, y) \mid x, y \in \mathbb{N}, a \le x \le b, \text{ and } g(x) \le y \le f(x)\}.$$

21. Prove that a set defined by (3.1) is countable if the basis elements in Step 1 are countable, the outside elements used in Step 2 are countable, and the rules specified in Step 2 are finite.

3.2 Recursive Functions and Procedures

Since we're going to be constructing functions and procedures in this section, we'd better agree on the idea of a procedure. From a computer science point of view a *procedure* is a program that performs one or more actions. So there is no requirement to return a specific value. For example, the execution of a statement like print(x, y) will cause the values of x and y to be printed. In this case, two actions are performed, and no values are returned. A procedure may also return one or more values through its argument list. For example, a statement like allocate(m, a, s) might perform the action of allocating a block of m memory cells and return the values a and s, where a is the beginning address of the block and the s tells whether the allocation was succesful.

Definition of Recursively Defined

A function or a procedure is said to be *recursively defined* if it is defined in terms of itself. In other words, a function f is recursively defined if at least one value $f(x)$ is defined in terms of another value $f(y)$, where $x \neq y$. Similarly, a procedure P is recursively defined if the actions of P for some argument x are defined in terms of the actions of P for another argument y, where $x \neq y$.

Many useful recursively defined functions have domains that are inductively defined sets. Similarly, many recursively defined procedures process elements from inductively defined sets. For these cases there are very useful construction techniques. Let's describe the two techniques.

Constructing a Recursively Defined Function **(3.6)**

If S is an inductively defined set, then we can construct a function f with domain S as follows:

1. For each basis element $x \in S$, specify a value for $f(x)$.
2. Give rules that, for any inductively defined element $x \in S$, will define $f(x)$ in terms of previously defined values of f.

Any function constructed by (3.6) is recursively defined because it is defined in terms of itself by the induction part of the definition. In a similar way we can construct a recursively defined procedure to process the elements of an inductively defined set.

Constructing a Recursively Defined Procedure **(3.7)**

If S is an inductively defined set, we can construct a procedure P to process the elements of S as follows:

1. For each basis element $x \in S$, specify a set of actions for $P(x)$.
2. Give rules that, for any inductively defined element $x \in S$, will define the actions of $P(x)$ in terms of previously defined actions of P.

In the following paragraphs we'll see how (3.6) and (3.7) can be used to construct recursively defined functions and procedures over a variety of inductively defined sets. Most of our examples will be functions. But we'll define a few procedures too.

3.2.1 Numbers

Let's see how some number functions can be defined recursively. To illustrate the idea, suppose we want to calculate the sum of the first n natural numbers for any $n \in \mathbb{N}$. Letting $f(n)$ denote the desired sum, we can write the informal definition

$$f(n) = 0 + 1 + 2 + \cdots + n.$$

We can observe, for example, that $f(0) = 0$, $f(1) = 1$, $f(2) = 3$, and so on. After a while we might notice that $f(3) = f(2) + 3 = 6$ and $f(4) = f(3) + 4 = 10$.

In other words, when $n > 0$, the definition can be transformed in the following way:

$$\begin{aligned} f(n) &= 0 + 1 + 2 + \cdots + n \\ &= (0 + 1 + 2 + \cdots + (n - 1)) + n \\ &= f(n - 1) + n. \end{aligned}$$

This gives us the recursive part of a definition of f for any $n > 0$. For the basis case we have $f(0) = 0$. So we can write the following recursive definition for f:

$$\begin{aligned} f(0) &= 0, \\ f(n) &= (n - 1) + n \quad \text{for } n > 0. \end{aligned}$$

There are two alternative forms that can be used to write a recursive definition. One form expresses the definition as an *if-then-else* equation. For example, f can be described in the following way.

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

Another form expresses the definition as equations whose left sides determine which equation to use in the evaluation of an expression rather than a conditional like $n > 0$. Such a form is called a *pattern-matching* definition because the equation chosen to evaluate $f(x)$ is determined uniquely by which left side $f(x)$ matches. For example, f can be described in the following way.

$$\begin{aligned} f(0) &= 1, \\ f(n + 1) &= f(n) + n + 1. \end{aligned}$$

For example, $f(3)$ matches $f(n + 1)$ with $n = 2$, so we would choose the second equation to evaluate $f(3) = f(2) + 3$, and so on.

A recursively defined function can be evaluated by a technique called *unfolding* the definition. For example, we'll evaluate the expression $f(4)$.

$$\begin{aligned} f(4) &= f(3) + 4 \\ &= f(2) + 3 + 4 \\ &= f(1) + 2 + 3 + 4 \\ &= f(0) + 1 + 2 + 3 + 4 \\ &= 0 + 1 + 2 + 3 + 4 \\ &= 10. \end{aligned}$$

example

3.16 Using the Floor Function

Let $f : \mathbb{N} \to \mathbb{N}$ be defined in terms of the floor function as follows:

$$\begin{aligned} &f(0) = 0, \\ &f(n) = f(\text{floor}(n/2)) + n \quad \text{for } n > 0. \end{aligned}$$

Notice in this case that $f(n)$ is not defined in terms of $f(n-1)$ but rather in terms of $f(\text{floor}(n/2))$. For example, $f(16) = f(8) + 16$. The first few values are $f(0) = 0$, $f(1) = 1$, $f(2) = 3$, $f(3) = 4$, and $f(4) = 7$. We'll calculate $f(25)$.

$$\begin{aligned} f(25) &= f(12) + 25 \\ &= f(6) + 12 + 25 \\ &= f(3) + 6 + 12 + 25 \\ &= f(1) + 3 + 6 + 12 + 25 \\ &= f(0) + 1 + 3 + 6 + 12 + 25 \\ &= 0 + 1 + 3 + 6 + 12 + 25 \\ &= 47. \end{aligned}$$

end example

example

3.17 Adding Odd Numbers

Let $f : \mathbb{N} \to \mathbb{N}$ denote the function to add up the first n odd natural numbers. So f has the following informal definition.

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

For example, the definition tells us that $f(0) = 1$. For $n > 0$ we can make the following transformation of $f(n)$ into an expression in terms of $f(n-1)$:

$$\begin{aligned} f(n) &= 1 + 3 + \cdots + (2n + 1) \\ &= (1 + 3 + \cdots + (2n - 1)) + (2n + 1) \\ &= (1 + 3 + \cdots + 2(n - 1) + 1) + (2n + 1) \\ &= f(n - 1) + 2n + 1. \end{aligned}$$

So we can make the following recursive definition of f:

$$\begin{aligned} &f(0) = 1, \\ &f(n) = f(n - 1) + 2n + 1 \quad \text{for } n > 0. \end{aligned}$$

Alternatively, we can write the recursive part of the definition as

$$f(n + 1) = f(n) + 2n + 3.$$

We can also write the defintion in the following if-then-else form.

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

Here is the evaluation of $f(3)$ using the if-then-else definition:

$$\begin{aligned} f(3) &= f(2) + 2(3) + 1 \\ &= f(1) + 2(2) + 1 + 2(3) + 1 \\ &= f(0) + 2(1) + 1 + 2(2) + 1 + 2(3) + 1 \\ &= 1 + 2(1) + 1 + 2(2) + 1 + 2(3) + 1 \\ &= 1 + 3 + 5 + 7 \\ &= 16. \end{aligned}$$

end example

example **3.18 The Rabbit Problem**

The *Fibonacci numbers* are the numbers in the sequence

$$0, 1, 1, 2, 3, 5, 8, 13, \ldots$$

where each number after the first two is computed by adding the preceding two numbers. These numbers are named after the mathematician Leonardo Fibonacci, who in 1202 introduced them in his book *Liber Abaci*, in which he proposed and solved the following problem: Starting with a pair of rabbits, how many pairs of rabbits can be produced from that pair in a year if it is assumed that every month each pair produces a new pair that becomes productive after one month?

For example, if we don't count the original pair and assume that the original pair needs one month to mature and that no rabbits die, then the number of new pairs produced each month for 12 consecutive months is given by the sequence

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89.$$

The sum of these numbers, which is 232, is the number of pairs of rabbits produced in one year from the original pair.

Fibonacci numbers seem to occur naturally in many unrelated problems. Of course, they can also be defined recursively. For example, letting fib(n) be the nth Fibonacci number, we can define fib recursively as follows:

$$\begin{aligned} &\text{fib}(0) = 0, \\ &\text{fib}(1) = 1, \\ &\text{fib}(n) = \text{fib}(n - 1) + \text{fib}(n - 2) \quad \text{for } n \geq 2. \end{aligned}$$

The third line could be written in pattern matching form as

$$\text{fib}(n + 2) = \text{fib}(n + 1) + \text{fib}(n).$$

The definition of fib in if-then-else form looks like

$$\begin{aligned}\text{fib}(n) = &\text{ if } n = 0 \text{ then } 0\\ &\text{ else if } n = 1 \text{ then } 1\\ &\text{ else fib}(n-1) + \text{fib}(n-2).\end{aligned}$$

end example

Sum and Product Notation

Many definitions and properties that we use without thinking are recursively defined. For example, given a sequence of numbers $(a_1, a_2, \ldots, a_n)$ we can represent the sum of the sequence with *summation notation* using the symbol Σ as follows.

$$\sum_{i=1}^{n} a_i = a_1 + a_2 + \cdots + a_n.$$

This notation has the following recursive definition, which makes the practical assumption that an empty sum is 0.

$$\sum_{i=1}^{n} a_i = \text{if } n = 0 \text{ then } 0 \text{ else } a_n + \sum_{i=1}^{n-1} a_i.$$

Similarly we can represent the product of the sequence with the following *product notation*, where the practical assumption is that an empty product is 1.

$$\prod_{i=1}^{n} a_i = \text{ if } n = 0 \text{ then } 1 \text{ else } a_n \cdot \prod_{i=1}^{n-1} a_i.$$

In the special case where $(a_1, a_2, \ldots, a_n) = (1, 2, \ldots, n)$ the product defines popular *factorial function*, which is denoted by $n!$ and is read "n factorial." In other words, we have

$$n! = (1)(2)\cdots(n-1)(n).$$

For example, $4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$, and $0! = 1$. So we can define $n!$ in the following recursive form.

$$n! = \text{if } n = 0 \text{ then } 1 \text{ else } n \cdot (n-1)!.$$

3.2.2 Strings

Let's see how some string functions can be defined recursively. To illustrate the idea, suppose we want to calculate the complement of any string over the alphabet $\{a, b\}$. For example, the complement of the string *bbab* is *aaba*.

Let $f(x)$ be the complement of x. To find a recursive definition for f we'll start by observing that an arbitrary string over $\{a, b\}$ is either Λ or has the form ay or by for some string y. So we'll define the result of f applied to each of these forms as follows:

$$\begin{aligned} f(\Lambda) &= \Lambda, \\ f(ax) &= bf(x), \\ f(bx) &= af(x). \end{aligned}$$

For example, we'll evaluate $f(bbab)$:

$$\begin{aligned} f(bbab) &= af(bab) \\ &= aaf(ab) \\ &= aabf(b) \\ &= aaba. \end{aligned}$$

Here are some more examples.

example 3.19 Prefixes of Strings

Consider the problem of finding the longest common prefix of two strings. A string p is a *prefix* of the string x if x can be written in the form $x = ps$ for some string s. For example, aab is the longest common prefix of the two strings $aabbab$ and $aababb$.

For two strings over $\{a, b\}$, let $f(x, y)$ be the longest common prefix of x and y. To find a recursive definition for f we can start by observing that an arbitrary string over $\{a, b\}$ is either the empty string Λ or has the form as or bs for some string s. In other words, the strings over $\{a, b\}$ are an inductively defined set. So we can define $f(s, t)$ by making sure that we take care to define it for all combinations of s and t. Here is a definition of f in pattern-matching form:

$$\begin{aligned} f(\Lambda, x) &= \Lambda, \\ f(x, \Lambda) &= \Lambda, \\ f(ax, by) &= \Lambda, \\ f(bx, ay) &= \Lambda, \\ f(ax, ay) &= af(x, y), \\ f(bx, by) &= bf(x, y). \end{aligned}$$

We can put the definition in if-then-else form as follows:

$$\begin{aligned} f(s,t) = {} & \text{if } s = \Lambda \text{ or } t = \Lambda \text{ then } \Lambda \\ & \text{else if } s = ax \text{ and } t = ay \text{ then } af(x,y) \\ & \text{else if } s = bx \text{ and } t = by \text{ then } bf(x,y) \\ & \text{else } \Lambda. \end{aligned}$$

We'll demonstrate the definition of f by calculating $f(aabbab,\ aababb)$:

$$\begin{aligned} f(aabbab, aababb) &= af(abbab, ababb) \\ &= aaf(bbab, babb) \\ &= aabf(bab, abb) \\ &= aab\Lambda \\ &= aab. \end{aligned}$$

end example

example 3.20 Converting Natural Numbers to Binary

Recall from Section 2.1 that we can represent a natural number x as

$$x = 2(\text{floor}(x/2)) + x \bmod 2.$$

This formula can be used to create a binary representation of x because x mod 2 is the rightmost bit of the representation. The next bit is found by computing floor($x/2$) mod 2. The next bit is floor(floor($x/2$)/2) mod 2, and so on. For example, we'll compute the binary representation of 13.

$$\begin{array}{rcll} 13 &=& 2\lfloor 13/2\rfloor + 13 \bmod 2 &=& 2(6)+1 \\ 6 &=& 2\lfloor 6/2\rfloor + 6 \bmod 2 &=& 2(3)+0 \\ 3 &=& 2\lfloor 3/2\rfloor + 3 \bmod 2 &=& 2(1)+1 \\ 1 &=& 2\lfloor 1/2\rfloor + 1 \bmod 2 &=& 2(0)+1 \end{array}$$

We can read off the remainders in reverse order to obtain 1101, which is the binary representation of 13.

Let's try to use this idea to write a recursive definition for the function "binary" to compute the binary representation for a natural number. If $x = 0$ or $x = 1$, then x is its own binary representation. If $x > 1$, then the binary representation of x is that of floor($x/2$) with the bit x mod 2 attached on the right end. So our recursive definition of binary can be written as follows, where "cat" is the string concatenation function.

$$\begin{aligned} &\text{binary}(0) = 0, \\ &\text{binary}(1) = 1, \\ &\text{binary}(x) = \text{cat}(\text{binary}(\lfloor x/2\rfloor), x \bmod 2) \quad \text{for } x > 1. \end{aligned} \tag{3.8}$$

The definition can be written in if-then-else form as

$$\begin{aligned} \text{binary}(x) = {} & \text{if } x = 0 \text{ or } x = 1 \text{ then } x \\ & \text{else cat}(\text{binary}(\text{floor}(x/2)), x \bmod 2). \end{aligned}$$

For example, we'll unfold the definition to calculate binary(13):

$$\begin{aligned}\text{binary}(13) &= \text{cat}(\text{binary}(6), 1)\\ &= \text{cat}(\text{cat}(\text{binary}(3), 0), 1)\\ &= \text{cat}(\text{cat}(\text{cat}(\text{binary}(1), 1), 0), 1)\\ &= \text{cat}(\text{cat}(\text{cat}(1, 1), 0), 1)\\ &= \text{cat}(\text{cat}(11, 0), 1)\\ &= \text{cat}(110, 1)\\ &= 1101.\end{aligned}$$

end example

3.2.3 Lists

Let's see how some functions that use lists can be defined recursively. To illustrate the idea, suppose we need to define the function $f : \mathbb{N} \to \text{lists}(\mathbb{N})$ that computes the following backwards sequence:

$$f(n) = \langle n, n-1, \ldots, 1, 0\rangle.$$

With a little help from the cons function for lists, we can transform the informal definition of $f(n)$ into a computable expression in terms of $f(n-1)$:

$$\begin{aligned}f(n) &= \langle n, n-1, \ldots, 1, 0\rangle\\ &= \text{cons}(n, \langle n-1, \ldots, 1, 0\rangle)\\ &= \text{cons}(n, f(n-1)).\end{aligned}$$

Therefore, f can be defined recursively by

$$\begin{aligned}f(0) &= \langle 0\rangle.\\ f(n) &= \text{cons}(n, f(n-1)) \text{ for } n > 0.\end{aligned}$$

This definition can be written in if-then-else form as

$$f(n) = \text{if } n = 0 \text{ then } \langle 0\rangle \text{ else cons}(n, f(n-1)).$$

To see how the evaluation works, look at the unfolding that results when we evaluate $f(3)$:

$$\begin{aligned}f(3) &= \text{cons}(3, f(2))\\ &= \text{cons}(3, \text{cons}(2, f(1)))\\ &= \text{cons}(3, \text{cons}(2, \text{cons}(1, f(0))))\\ &= \text{cons}(3, \text{cons}(2, \text{cons}(1, \langle 0\rangle)))\\ &= \text{cons}(3, \text{cons}(2, \langle 1, 0\rangle))\\ &= \text{cons}(3, \langle 2, 1, 0\rangle)\\ &= \langle 3, 2, 1, 0\rangle.\end{aligned}$$

We haven't given a recursively defined procedure yet. So let's give one for the problem we've been discussing. For example, suppose that $P(n)$ prints out the numbers in the list $\langle n, n-1, \ldots, 0\rangle$. A recursive definition of P can be written as follows.

```
P(n):   if $n = 0$ then print(0)
        else
            print($n$);
            $P(n-1)$
        fi.
```

example

3.21 Length of a List

Let S be a set and let "length" be the function of type lists(S) $\to \mathbb{N}$, which returns the number of elements in a list. We can define length recursively by noticing that the length of an empty list is zero and the length of a nonempty list is one plus the length of its tail. A definition follows.

$$\begin{aligned} \text{length}\,(\langle\ \rangle) &= 0, \\ \text{length}\,(\text{cons}\,(x, t)) &= 1 + \text{length}\,(t)\,. \end{aligned}$$

Recall that the infix form of cons(x, t) is $x :: t$. So we could just as well write the second equation as

$$\text{length}(x :: t) = 1 + \text{length}(t).$$

Also, we could write the recursive part of the definition with a condition as follows:

$$\text{length}\,(L) = 1 + \text{length}\,(\text{tail}\,(L)) \quad \text{for } L \neq \langle\ \rangle\,.$$

In if-then-else form the definition can be written as follows:

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

The length function can be evaluated by unfolding its definition. For example, we'll evaluate length($\langle a, b, c\rangle$).

$$\begin{aligned} \text{length}\,(\langle a, b, c\rangle) &= 1 + \text{length}\,(\langle b, c\rangle) \\ &= 1 + 1 + \text{length}\,(\langle c\rangle) \\ &= 1 + 1 + 1 + \text{length}\,(\langle\ \rangle) \\ &= 1 + 1 + 1 + 0 \\ &= 3. \end{aligned}$$

end example

example

3.22 The Distribute Function

Suppose we want to write a recursive definition for the distribute function, which we'll denote by "dist." Recall, for example, that

$$\text{dist}(a,\ \langle b,\ c,\ d,\ e\rangle) = \langle (a,\ b),\ (a,\ c),\ (a,\ d),\ (a,\ e)\rangle.$$

To discover the recursive part of the definition, we'll rewrite the example equation by splitting up the lists into head and tail components as follows:

$$\begin{aligned}\text{dist}\,(a, \langle b, c, d, e\rangle) &= \langle (a,b), (a,c), (a,d), (a,e)\rangle \\ &= (a,b) :: \langle (a,c), (a,d), (a,e)\rangle \\ &= (a,b) :: \text{dist}(a, \langle c, d, e\rangle).\end{aligned}$$

That's the key to the recursive part of the definition. Since we are working with lists, the basis case is dist(a, $\langle\ \rangle$), which we define as $\langle\ \rangle$. So the recursive definition can be written as follows:

$$\begin{aligned}\text{dist}(x, \langle\ \rangle) &= \langle\ \rangle, \\ \text{dist}(x, h :: T) &= (x,h) :: \text{dist}\,(x, T)\,.\end{aligned}$$

For example, we'll evaluate the expression dist(3, $\langle 10,\ 20\rangle$):

$$\begin{aligned}\text{dist}\,(3, \langle 10, 20\rangle) &= (3,10) :: \text{dist}\,(3, \langle 20\rangle) \\ &= (3,10) :: (3,20) :: \text{dist}\,(3, \langle\ \rangle) \\ &= (3,10) :: (3,20) :: \langle\ \rangle \\ &= (3,10) :: \langle (3,20)\rangle \\ &= \langle (3,10)\,, (3,20)\rangle\,.\end{aligned}$$

An if-then-else difinition of dist takes the following form:

$$\begin{aligned}\text{dist}(x, L) = {} & \text{if } L = \langle\ \rangle \text{ then } \langle\ \rangle \\ & \text{else } (x, \text{head}(L)) :: \text{dist}(x, \text{tail}(L)).\end{aligned}$$

end example

example

3.23 The Pairs Function

Recall that the "pairs" function creates a list of pairs of corresponding elements from two lists. For example,

$$\text{pairs}(\langle a,\ b,\ c\rangle,\ \langle 1,\ 2,\ 3\rangle) = \langle (a,\ 1),\ (b,\ 2),\ (c,\ 3)\rangle.$$

To discover the recursive part of the definition, we'll rewrite the example equation by splitting up the lists into head and tail components as follows:

$$\begin{aligned}\text{pairs}(\langle a, b, c\rangle, \langle 1, 2, 3\rangle) &= \langle (a,1), (b,2), (c,3)\rangle \\ &= (a,1) :: \langle (b,2), (c,3)\rangle \\ &= (a,1) :: \text{pairs}(\langle b, c\rangle, \langle 2, 3\rangle).\end{aligned}$$

Now the pairs function can be defined recursively by the following equations:

$$\begin{aligned}\text{pairs}(\langle\ \rangle, \langle\ \rangle) &= \langle\ \rangle,\\ \text{pairs}(x :: T, y :: U) &= (x, y) :: \text{pairs}(T, U).\end{aligned}$$

For example, we'll evaluate the expression pairs($\langle a, b\rangle$, $\langle 1, 2\rangle$):

$$\begin{aligned}\text{pairs}(\langle a, b\rangle, \langle 1, 2\rangle) &= (a, 1) :: \text{pairs}(\langle b\rangle, \langle 1\rangle)\\ &= (a, 1) :: (b, 2) :: \text{pairs}(\langle\ \rangle, \langle\ \rangle)\\ &= (a, 1) :: (b, 2) :: \langle\ \rangle\\ &= (a, 1) :: \langle (b, 2)\rangle\\ &= \langle (a, 1), (b, 2)\rangle.\end{aligned}$$

end example

example

3.24 The ConsRight Function

Suppose we need to give a recursive definition for the sequence function. Recall, for example, that seq(4) = $\langle 0, 1, 2, 3, 4\rangle$. Good old "cons" doesn't seem up to the task. For example, if we somehow have computed seq(3), then cons(4, seq(3)) = $\langle 4, 0, 1, 2, 3\rangle$. It would be nice if we had a constructor to place an element on the right of a list, just as cons places an element on the left of a list. We'll write a definition for the function "consR" to do just that. For example, we want

$$\text{consR}(\langle a, b, c\rangle, d) = \langle a, b, c, d\rangle.$$

We can get an idea of how to proceed by rewriting the above equation as follows in terms of the infix form of cons:

$$\begin{aligned}\text{consR}(\langle a, b, c\rangle, d) &= \langle a, b, c, d\rangle\\ &= a :: \langle b, c, d\rangle\\ &= a :: \text{consR}(\langle b, c\rangle, d).\end{aligned}$$

So the clue is to split the list $\langle a, b, c\rangle$ into its head and tail. We can write the recursive definition of consR using the if-then-else form as follows:

$$\begin{aligned}\text{consR}(L, a) = \ &\text{if } L = \langle\ \rangle \text{ then } \langle a\rangle\\ &\text{else head}(L) :: \text{consR}(\text{tail}(L), a).\end{aligned}$$

This definition can be written in pattern-matching form as follows:

$$\begin{aligned}\text{consR}(\langle\ \rangle, a) &= a :: \langle\ \rangle,\\ \text{consR}(b :: T, a) &= b :: \text{consR}(T, a).\end{aligned}$$

For example, we can construct the list $\langle x, y\rangle$ with consR as follows:

$$\begin{aligned}\text{consR}\left(\text{consR}\left(\langle\ \rangle, x\right), y\right) &= \text{consR}\left(x :: \langle\ \rangle, y\right)\\ &= x :: \text{consR}\left(\langle\ \rangle, y\right)\\ &= x :: y :: \langle\ \rangle\\ &= x :: \langle y\rangle\\ &= \langle x, y\rangle .\end{aligned}$$

end example

example **3.25 Concatenation of Lists**

An important operation on lists is the concatenation of two lists into a single list. Let "cat" denote the concatenation function. Its type is lists(A) $\times$ lists(A) $\to$ lists(A). For example,

$$\text{cat}(\langle a,\ b\rangle,\ \langle c,\ d\rangle) = \langle a,\ b,\ c,\ d\rangle.$$

Since both arguments are lists, we have some choices to make. Notice, for example, that we can rewrite the equation as follows:

$$\begin{aligned}\text{cat}(\langle a,b\rangle,\langle c,d\rangle) &= \langle a,b,c,d\rangle\\ &= a :: \langle b,c,d\rangle\\ &= a :: \text{cat}(\langle b\rangle,\langle c,d\rangle)\end{aligned}$$

So the recursive part can be written in terms of the head and tail of the first argument list. Here's an if-then-else definition for cat.

$$\begin{aligned}\text{cat}(L,M) = \text{if } L = \langle\ \rangle &\text{ then } M\\ &\text{else head}(L) :: \text{cat}(\text{tail}(L),M).\end{aligned}$$

We'll unfold the definition to evaluate the expression cat($\langle a,\ b\rangle$, $\langle c,\ d\rangle$):

$$\begin{aligned}\text{cat}\left(\langle a,b\rangle, \langle c,d\rangle\right) &= a :: \text{cat}\left(\langle b\rangle, \langle c,d\rangle\right)\\ &= a :: b :: \text{cat}\left(\langle\ \rangle, \langle c,d\rangle\right)\\ &= a :: b :: \langle c,d\rangle\\ &= a :: \langle b,c,d\rangle\\ &= \langle a,b,c,d\rangle .\end{aligned}$$

We can also write cat as a recursively defined procedure that prints out the elements of the two lists:

```
cat($K$, $L$):  **if** $K = \langle\ \rangle$ **then** print($L$)
             **else**
                print(head($K$));
                cat(tail($K$), $L$)
             **fi.**
```

end example

example

3.26 Sorting a List by Insertion

Let's define a function to sort a list of numbers by repeatedly inserting a new number into an already sorted list of numbers. Suppose "insert" is a function that does this job. Then the sort function itself is easy. For a basis case, notice that the empty list is already sorted. For the recursive case we sort the list $x :: L$ by inserting x into the list obtained by sorting L. The definition can be written as follows:

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

Everything seems to make sense as long as insert does its job. We'll assume that whenever the number to be inserted is already in the list, then a new copy will be placed to the left of the one already there. Now let's define insert. Again, the basis case is easy. The empty list is sorted, and to insert x into $\langle\ \rangle$, we simply create the singleton list $\langle x\rangle$. Otherwise—if the sorted list is not empty—either x belongs on the left of the list, or it should actually be inserted somewhere else in the list. An if-then-else definition can be written as follows:

$$\begin{aligned}\text{insert}(x, S) = {} & \text{if } S = \langle\ \rangle \text{ then } \langle x\rangle\\ & \text{else if } x \leq \text{ head}(S) \text{ then } x :: S\\ & \text{else head}(S) :: \text{insert}(x, \text{tail}(S)).\end{aligned}$$

Notice that insert works only when S is already sorted. For example, we'll unfold the definition of insert(3, ⟨1, 2, 6, 8⟩):

$$\begin{aligned}\text{insert}(3, \langle 1,2,6,8\rangle) &= 1 :: \text{insert}(3, \langle 2,6,8\rangle)\\ &= 1 :: 2 :: \text{insert}(3, \langle 6,8\rangle)\\ &= 1 :: 2 :: 3 :: \langle 6,8\rangle\\ &= \langle 1,2,3,6,8\rangle.\end{aligned}$$

end example

example

3.27 The Map Function

We'll construct a recursive definition for the map function. Recall, for example that

$$\text{map}(f, \langle a,b,c\rangle) = \langle f(a), f(b), f(c)\rangle.$$

Since the second argument is a list, it makes sense to define the basis case as $\text{map}(f, \langle\ \rangle) = \langle\ \rangle$. To discover the recursive part of the definition, we'll rewrite the example equation as follows:

$$\begin{aligned}\text{map}(f, \langle a,b,c\rangle) &= \langle f(a), f(b), f(c)\rangle\\ &= f(a) :: \langle f(b), f(c)\rangle\\ &= f(a) :: \text{map}(f, \langle b,c\rangle).\end{aligned}$$

So the recursive part can be written in terms of the head and tail of the input list. Here's an if-then-else definition for map.

$$\begin{aligned}\text{map}(f, L) = {} & \text{if } L = \langle\ \rangle \text{ then } \langle\ \rangle \\ & \text{else } f(\text{head}(L)) :: \text{map}(f, \text{tail}(L)).\end{aligned}$$

For example, we'll evaluate the expression map(f, $\langle a, b, c\rangle$).

$$\begin{aligned}\text{map}(f, \langle a,b,c\rangle) &= f(a) :: \text{map}(f, \langle b,c\rangle) \\ &= f(a) :: f(b) :: \text{map}(f, \langle c\rangle) \\ &= f(a) :: f(b) :: f(c) :: \text{map}(f, \langle\ \rangle) \\ &= f(a) :: f(b) :: f(c) :: \langle\ \rangle \\ &= \langle f(a), f(b), f(c)\rangle.\end{aligned}$$

end example

3.2.4 Binary Trees

Let's look at some functions that compute properties of binary trees. To start, suppose we need to know the number of nodes in a binary tree. Since the set of binary trees over a particular set can be defined inductively, we should be able to come up with a recursively defined function that suits our needs. Let "nodes" be the function that returns the number of nodes in a binary tree. Since the empty tree has no nodes, we have nodes($\langle\ \rangle$) = 0. If the tree is not empty, then the number of nodes can be computed by adding 1 to the number of nodes in the left and right subtrees. The equational definition of nodes can be written as follows:

$$\begin{aligned}\text{nodes}(\langle\ \rangle) &= 0, \\ \text{nodes}(\text{tree}(L, a, R)) &= 1 + \text{nodes}(L) + \text{nodes}(R).\end{aligned}$$

If we want the corresponding if-then-else form of the definition, it looks like

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

For example, we'll evaluate nodes(T) for $T = \langle\langle\langle\ \rangle, a, \langle\ \rangle\ \rangle, b, \langle\ \rangle\rangle$:

$$\begin{aligned}\text{nodes}(T) &= 1 + \text{nodes}(\langle\langle\ \rangle, a, \langle\ \rangle\rangle) + \text{nodes}(\langle\ \rangle) \\ &= 1 + 1 + \text{nodes}(\langle\ \rangle) + \text{nodes}(\langle\ \rangle) + \text{nodes}(\langle\ \rangle) \\ &= 1 + 1 + 0 + 0 + 0 \\ &= 2.\end{aligned}$$

example **3.28 A Binary Search Tree**

Suppose we have a binary search tree whose nodes are numbers, and we want to add a new number to the tree, under the assumption that the new tree is still a binary search tree. A function to do the job needs two arguments, a number x and a binary search tree T. Let the name of the function be "insert."

The basis case is easy. If $T = \langle\ \rangle$, then return tree($\langle\ \rangle$, x, $\langle\ \rangle$). The recursive part is straightforward. If $x <$ root(T), then we need to replace the subtree left(T) by insert(x, left(T)). Otherwise, we replace right(T) by insert(x, right(T)). Notice that repeated elements are entered to the right. If we didn't want to add repeated elements, then we could simply return T whenever $x =$ root(T). The if-then-else form of the definition is

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

Now suppose we want to build a binary search tree from a given list of numbers in which the numbers are in no particular order. We can use the insert function as the main ingredient in a recursive definition. Let "makeTree" be the name of the function. We'll use two variables to describe the function, a binary search tree T and a list of numbers L.

$$\begin{aligned}\text{makeTree}(T,L) = {} & \text{if } L = \langle\ \rangle \text{ then } T\\ & \text{else makeTree}(\text{insert}(\text{head}(L), T), \text{tail}(L)).\end{aligned} \tag{3.9}$$

To construct a binary search tree with this function, we apply makeTree to the pair of arguments ($\langle\ \rangle$, L). As an example, the reader should unfold the definition for the call makeTree($\langle\ \rangle$, $\langle 3, 2, 4\rangle$).

The function makeTree can be defined another way. Suppose we consider the following definition for constructing a binary search tree:

$$\begin{aligned}\text{makeTree}(T,L) = {} & \text{if } L = \langle\ \rangle \text{ then } T\\ & \text{else insert}(\text{head}(L), \text{makeTree}(T, \text{tail}(L))).\end{aligned} \tag{3.10}$$

You should evaluate the expression makeTree($\langle\ \rangle$, $\langle 3, 2, 4\rangle$) by unfolding this alternative definition. It should help explain the difference between the two definitions.

end example

Traversing Binary Trees

There are several useful ways to list the nodes of a binary tree. The three most popular methods of traversing a binary tree are called *preorder, inorder,* and *postorder.* We'll start with the definition of a preorder traversal.

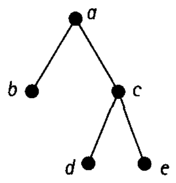


Figure 3.6 A binary tree.

Preorder Traversal

The *preorder* traversal of a binary tree starts by visiting the root. Then there is a preorder traversal of the left subtree followed by a preorder traversal of the right subtree.

For example, the preorder listing of the nodes of the binary tree in Figure 3.6 is $\langle a, b, c, d, e\rangle$. It's common practice to write the listing without any punctuation symbols as

$$a\ b\ c\ d\ e.$$

example **3.29 A Preorder Procedure**

Since binary trees are inductively defined, we can easily write a recursively defined procedure to output the preorder listing of a binary tree. For example, the following recursively defined procedure prints the preorder listing of its argument T.

```
Preorder($T$):  if $T \neq \langle\,\rangle$ then
                   print(root($T$));
                   Preorder(left($T$));
                   Preorder(right($T$))
                fi.
```

end example

example **3.30 A Preorder Function**

Now let's write a function to compute the preorder listing of a binary tree. Letting "preOrd" be the name of the preorder function, an equational definition can be written as follows:

$$\text{preOrd}\,(\langle\,\rangle) = \langle\,\rangle,$$
$$\text{preOrd}\,(\text{tree}\,(L, x, R)) = x :: \text{cat}\,(\text{preOrd}\,(L)\,, \text{preOrd}\,(R))\,.$$

The if-then-else form of preOrd can be written as follows:

$$\begin{aligned}\text{preOrd}(T) = {} & \text{if } T = \langle\ \rangle \text{ then } \langle\ \rangle \\ & \text{else root}(T) :: \text{cat}(\text{preOrd}(\text{left}(T)), \text{preOrd}(\text{right}(T))).\end{aligned}$$

We'll evaluate the expression preOrd(T) for the tree $T = \langle\langle\langle\ \rangle, a, \langle\ \rangle\rangle, b, \langle\ \rangle\rangle$:

$$\begin{aligned}\text{preOrd}(T) &= b :: \text{cat}(\text{preOrd}(\langle\langle\ \rangle, a\langle\ \rangle\rangle), \text{preOrd}(\langle\ \rangle)) \\ &= b :: \text{cat}(a :: \text{cat}(\text{preOrd}(\langle\ \rangle), \text{preOrd}(\langle\ \rangle)), \text{preOrd}(\langle\ \rangle)) \\ &= b :: \text{cat}(a :: \langle\ \rangle, \langle\ \rangle) \\ &= b :: \text{cat}(\langle a\rangle, \langle\ \rangle) \\ &= b :: \langle a\rangle \\ &= \langle b, a\rangle.\end{aligned}$$

end example

The definitions for the inorder and postorder traversals of a binary tree are similar to the preorder traversal. The only difference is when the root is visited during the traversal.

Inorder Traversal

The *inorder* traversal of a binary tree starts with an inorder traversal of the left subtree. Then the root is visited. Lastly, there is an inorder traversal of the right subtree.

For example, the inorder listing of the tree in Figure 3.6 is

$$b\ a\ d\ c\ e.$$

Postorder Traversal

The *postorder* traversal of a binary tree starts with a postorder traversal of the left subtree and is followed by a postorder traversal of the right subtree. Lastly, the root is visited.

The postorder listing of the tree in Figure 3.6 is

$$b\ d\ e\ c\ a.$$

We'll leave the construction of the inorder and postorder procedures and functions as exercises.

3.2.5 Two More Problems

We'll look at two more problems, each of which requires a little extra thinking on the way to a solution.

The Repeated Element Problem

Suppose we want to remove repeated elements from a list. Depending on how we proceed, there might be different solutions. For example, we can remove the repeated elements from the list $\langle u, g, u, h, u\rangle$ in three ways, depending on which occurrence of u we keep: $\langle u, g, h\rangle$, $\langle g, u, h\rangle$, or $\langle g, h, u\rangle$. We'll solve the problem by always keeping the leftmost occurrence of each element. Let "remove" be the function that takes a list L and returns the list remove(L), which has no repeated elements and contains the leftmost occurrence of each element of L.

To start things off, we can say remove($\langle\ \rangle$) = $\langle\ \rangle$. Now if $L \neq \langle\ \rangle$, then L has the form $L = b :: M$ for some list M. In this case, the head of remove(L) should be b. The tail of remove(L) can be obtained by removing all occurrences of b from M and then removing all repeated elements from the resulting list. So we need a new function to remove all occurrences of an element from a list.

Let removeAll(b, M) denote the list obtained from M by removing all occurrences of b. Now we can write an equational definition for the remove function as follows:

$$\begin{aligned}&\text{remove}(\langle\ \rangle) = \langle\ \rangle,\\ &\text{remove}(b :: M) = b :: \text{remove}(\text{removeAll}(b, M)).\end{aligned}$$

We can rewrite the solution in if-then-else form as follows:

$$\begin{aligned}\text{remove}(L) = {} & \text{if } L = \langle\ \rangle \text{ then } \langle\ \rangle\\ & \text{else head}(L) :: \text{remove}(\text{removeAll}(\text{head}(L), \text{tail}(L))).\end{aligned}$$

To complete the task, we need to define the "removeAll" function. The basis case is removeAll(b, $\langle\ \rangle$) = $\langle\ \rangle$. If $M \neq \langle\ \rangle$, then the value of removeAll(b, M) depends on head(M). If head(M) = b, then throw it away and return the value of removeAll(b, tail(M)). But if head(M) $\neq b$, then it's a keeper. So we should return the value head(M) :: removeAll(b, tail(M)). We can write the definition in if-then-else form as follows:

$$\begin{aligned}\text{removeAll}(b, M) = {} & \text{if } M = \langle\ \rangle \text{ then } \langle\ \rangle\\ & \text{else if head}(M) = b \text{ then}\\ & \quad \text{removeAll}(b, \text{tail}(M))\\ & \text{else}\\ & \quad \text{head}(M) :: \text{removeAll}(b, \text{tail}(M)).\end{aligned}$$

We'll evaluate the expression removeAll(b, $\langle a,\ b,\ c,\ b\rangle$):

$$\begin{aligned}\text{removeAll}(b, \langle a, b, c, b\rangle) &= a :: \text{removeAll}(b, \langle b, c, b\rangle)\\ &= a :: \text{removeAll}(b, \langle c, b\rangle)\\ &= a :: c :: \text{removeAll}(b, \langle b\rangle)\\ &= a :: c :: \text{removeAll}(b, \langle\ \rangle)\\ &= a :: c :: \langle\ \rangle\\ &= a :: \langle c\rangle\\ &= \langle a, c\rangle .\end{aligned}$$

Try to write out each unfolding step in the evaluation of the expression remove($\langle b,\ a,\ b\rangle$). Be sure to start writing at the left-hand edge of your paper.

The Power Set Problem

Suppose we want to construct the power set of a finite set. One solution uses the fact that power($\{x\} \cup T$) is the union of power(T) and the set obtained from power(T) by adding x to each of its elements. Let's see whether we can discover a solution technique by considering a small example. Let $S = \{a,\ b,\ c\}$. Then we can write power(S) as follows:

$$\begin{aligned}\text{power}(S) &= \{\{\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}\\ &= \{\{\}, \{b\}, \{c\}, \{b, c\}\} \cup \{\{\underline{a}\}, \{\underline{a}, b\}, \{\underline{a}, c\}, \{\underline{a}, b, c\}\} .\end{aligned}$$

We've written power(S) $= A \cup B$, where B is obtained from A by adding the underlined element $\underline{a}$ to each set in A. If we represent S as the list $\langle a,\ b,\ c\rangle$, then we can restate the definition for power(S) as the concatenation of the following two lists:

$$\langle\langle\ \rangle, \langle b\rangle, \langle c\rangle, \langle b,\ c\rangle\rangle \quad \text{and} \quad \langle\langle a\rangle, \langle a,\ b\rangle, \langle a,\ c\rangle, \langle a,\ b,\ c\rangle\rangle.$$

The first of these lists is power($\langle b,\ c\rangle$). The second list can be obtained from power($\langle b,\ c\rangle$) by working backward to the answer as follows:

$$\begin{aligned}\langle\langle a\rangle, \langle a, b\rangle, \langle a, c\rangle, \langle a, b, c\rangle\rangle &= \langle a :: \langle\ \rangle, a :: \langle b\rangle, a :: \langle c\rangle, a :: \langle b, c\rangle\rangle\\ &= \text{map}(::, \langle\langle a, \langle\ \rangle\rangle, \langle a, \langle b\rangle\rangle, \langle a, \langle c\rangle\rangle, \langle a, \langle b, c\rangle\rangle\rangle)\\ &= \text{map}(::, \text{dist}(a, \text{power}(\langle b, c\rangle))) .\end{aligned}$$

This example is the key to the recursive part of the definition. Using the fact that power($\langle\ \rangle$) $= \langle\langle\ \rangle\rangle$ as the basis case, we can write down the following definition for power:

$$\begin{aligned}\text{power}(\langle\ \rangle) &= \langle\langle\ \rangle\rangle ,\\ \text{power}(a :: L) &= \text{cat}(\text{power}(L), \text{map}(::, \text{dist}(a, \text{power}(L)))) .\end{aligned}$$

The if-then-else form of the definition can be written as follows:

$$\begin{aligned}\text{power}(L) = \text{if } L = \langle\,\rangle \text{ then } \langle\langle\,\rangle\rangle \text{ else}\\ \text{cat}(\text{power}(\text{tail}(L)), \text{map}(::, \text{dist}(\text{head}(L), \text{power}(\text{tail}(L))))).\end{aligned}$$

We'll evaluate the expression power($\langle a, b\rangle$). The first step yields the equation

$$\text{power}(\langle a, b\rangle) = \text{cat}(\text{power}(\langle b\rangle), \text{map}(::, \text{dist}(a, \text{power}(\langle b\rangle)))).$$

Now we'll evaluate power($\langle b\rangle$) and substitute it in the preceding equation:

$$\begin{aligned}\text{power}(\langle b\rangle) &= \text{cat}(\text{power}(\langle\,\rangle), \text{map}(::, \text{dist}(b, \text{power}(\langle\,\rangle))))\\ &= \text{cat}(\langle\langle\,\rangle\rangle, \text{map}(::, \text{dist}(b, \langle\langle\,\rangle\rangle)))\\ &= \text{cat}(\langle\langle\,\rangle\rangle, \text{map}(::, \langle\langle b, \langle\,\rangle\rangle\rangle))\\ &= \text{cat}(\langle\langle\,\rangle\rangle, \langle b :: \langle\,\rangle\rangle)\\ &= \text{cat}(\langle\langle\,\rangle\rangle, \langle\langle b\rangle\rangle)\\ &= \langle\langle\,\rangle, \langle b\rangle\rangle.\end{aligned}$$

Now we can continue with the evaluation of power($\langle a, b\rangle$):

$$\begin{aligned}\text{power}(\langle a, b\rangle) &= \text{cat}(\text{power}(\langle b\rangle), \text{map}(::, \text{dist}(a, \text{power}(\langle b\rangle))))\\ &= \text{cat}(\langle\langle\,\rangle, \langle b\rangle\rangle, \text{map}(::, \text{dist}(a, \langle\langle\,\rangle, \langle b\rangle\rangle)))\\ &= \text{cat}(\langle\langle\,\rangle, \langle b\rangle\rangle, \text{map}(::, \langle\langle a, \langle\,\rangle\rangle, \langle a, \langle b\rangle\rangle\rangle))\\ &= \text{cat}(\langle\langle\,\rangle, \langle b\rangle\rangle, \langle a :: \langle\,\rangle, a :: \langle b\rangle\rangle)\\ &= \text{cat}(\langle\langle\,\rangle, \langle b\rangle\rangle, \langle\langle a\rangle, \langle a, b\rangle\rangle)\\ &= \langle\langle\,\rangle, \langle b\rangle, \langle a\rangle, \langle a, b\rangle\rangle.\end{aligned}$$

3.2.6 Infinite Sequences

Let's see how some infinite sequences can be defined recursively. To illustrate the idea, suppose the function "ints" returns the following infinite sequence for any integer x:

$$\text{ints}(x) = \langle x, x + 1, x + 2, \ldots\rangle.$$

We'll assume that that the list operations of cons, head, and tail work for infinite sequences. For example, the following relationships hold.

$$\begin{aligned}&\text{ints}(x) = x :: \text{ints}(x + 1),\\ &\text{head}(\text{ints}(x)) = x,\\ &\text{tail}(\text{ints}(x)) = \text{ints}(x + 1).\end{aligned}$$

Even though the definition of ints does not conform to (3.6), it is still recursively defined because it is defined in terms of itself. If we executed the definition, an infinite loop would construct the infinite sequence. For example, ints(0) would construct the infinite sequence of natural numbers as follows:

$$\begin{aligned}\text{ints}(0) &= 0 :: \text{ints}(1)\\ &= 0 :: 1 :: \text{ints}(2)\\ &= 0 :: 1 :: 2 :: \text{ints}(3)\\ &= \ldots .\end{aligned}$$

In practice, an infinite sequence is used as an argument and is is evaluated only when some of its values are needed. Once the needed values are computed, the evaluation stops. This is an example of a technique called *lazy evaluation.* For example, the following function returns the nth element of an infinite sequence s.

$$\text{get}(n, s) = \text{if } n = 1 \text{ then head}(s) \text{ else get}(n - 1, \text{tail}(s)).$$

example

3.31 Picking Elements

We'll get the third element from the infinite sequence ints(6) by unfolding the expression get(3, ints(6)).

$$\begin{aligned}\text{get}(3, \text{ints}(6)) &= \text{get}(2, \text{tail}(\text{ints}(6)))\\ &= \text{get}(1, \text{tail}(\text{tail}(\text{ints}(6))))\\ &= \text{head}(\text{tail}(\text{tail}(\text{ints}(6))))\\ &= \text{head}(\text{tail}(\text{tail}(6 :: \text{ints}(7))))\\ &= \text{head}(\text{tail}(\text{ints}(7)))\\ &= \text{head}(\text{tail}(7 :: \text{ints}(8)))\\ &= \text{head}(\text{ints}(8))\\ &= \text{head}(8 :: \text{ints}(9))\\ &= 8.\end{aligned}$$

end example

example

3.32 Summing

Suppose we need a function to sum the first n elements in an infinite sequence s of integers. The following definition does the trick:

$$\text{sum}(n, s) = \text{if } n = 0 \text{ then } 0 \text{ else head}(s) + \text{sum}(n - 1, \text{tail}(s)).$$

We'll compute the sum of the first three numbers in ints(4):

$$\begin{aligned}\text{sum}\,(3, \text{ints}\,(4)) &= 4 + \text{sum}\,(2, \text{ints}\,(5)) \\ &= 4 + 5 + \text{sum}\,(1, \text{ints}\,(6)) \\ &= 4 + 5 + 6 + \text{sum}\,(0, \text{ints}\,(7)) \\ &= 4 + 5 + 6 + 0 \\ &= 15.\end{aligned}$$

end example

example

3.33 The Sieve of Eratosthenes

Suppose we want to study prime numbers. For example, we might want to find the 500th prime, we might want to find the difference between the 500th and 501st primes, and so on. One way to proceed might be to define functions to extract information from the following infinite sequence of all prime numbers.

$$Primes = \langle 2,\ 3,\ 5,\ 7,\ 11,\ 13,\ 17,\ \ldots\rangle.$$

We'll construct this infinite sequence by the method of Eratosthenes (called *the sieve of Eratosthenes*). The method starts with the infinite sequence ints(2):

$$\text{ints}(2) = \langle 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10,\ \ldots\rangle.$$

The next step removes all multiples of 2 (except 2) to obtain the infinite sequence

$$\langle 2,\ 3,\ 5,\ 7,\ 9,\ 11,\ 13,\ 15,\ \ldots\rangle.$$

The next step removes all multiples of 3 (except 3) to obtain the infinite sequence

$$\langle 2,\ 3,\ 5,\ 7,\ 11,\ 13,\ 17,\ \ldots\rangle.$$

The process continues in this way.

We can construct the desired infinite sequence of primes once we have the function to remove multiples of a number from an infinite sequence. If we let remove(n, s) denote the infinite sequence obtained from s by removing all multiples of n, then we can define the sieve process as follows for an infinite sequence s of numbers:

$$\text{sieve}(s) = \text{head}(s) :: \text{sieve}(\text{remove}(\text{head}(s),\ \text{tail}(s))).$$

But we need to define the remove function. Notice that for natural numbers m and n with $n > 0$ that we have the following equivalences:

$$m \text{ is a multiple of } n \text{ iff } n \text{ divides } m \text{ iff } m \bmod n = 0.$$

This allows us to write the following definition for the remove function:

$$\begin{aligned}\text{remove}(n, s) = {} & \text{if head}(s) \bmod n = 0 \text{ then remove}(n, \text{tail}(s)) \\ & \text{else head}(s) :: \text{remove}(n, \text{tail}(s)).\end{aligned}$$

Then our desired sequence of primes is represented by the expression

$$\mathit{Primes} = \text{sieve}(\text{ints}(2)).$$

In the exercises we'll evaluate some functions dealing with primes.

end example

Exercises

Evaluating Recursively Defined Functions

1. Given the following definition for the nth Fibonacci number:

$$\begin{aligned}&\text{fib}(0) = 0,\\ &\text{fib}(1) = 1,\\ &\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2) \quad \text{if } n > 1.\end{aligned}$$

   Write down each step in the evaluation of fib(4).

2. Given the following definition for the length of a list:

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

   Write down each step in the evaluation of length($\langle r, s, t, u\rangle$).

3. For each of the two definitions of "makeTree" given by (3.9) and (3.10), write down all steps to evaluate makeTree($\langle\ \rangle$, $\langle 3, 2, 4\rangle$).

Numbers

4. Construct a recursive definition for each of the following functions, where all variables are natural numbers.
   a. $f(n) = 0 + 2 + 4 + \cdots + 2n$.
   b. $f(n) = \text{floor}(0/2) + \text{floor}(1/2) + \cdots + \text{floor}(n/2)$.
   c. $f(k,n) = \gcd(1, n) + \gcd(2, n) + \cdots + \gcd(k, n)$ for $k > 0$.
   d. $f(n) = (0 \bmod 2) + (1 \bmod 3) + \cdots + (n \bmod (n + 2))$.
   e. $f(n, k) = 0 + k + 2k + \cdots + nk$.
   f. $f(n, k) = k + (k + 1) + (k + 2) + \cdots + (k + n)$.

Strings

5. Construct a recursive definition for each of the following string functions for strings over the alphabet $\{a, b\}$.

   a. $f(x)$ returns the reverse of x.
   b. $f(x) = xy$, where y is the reverse of x.
   c. $f(x, y)$ tests whether x is a prefix of y.
   d. $f(x, y)$ tests whether $x = y$.
   e. $f(x)$ tests whether x is a palindrome.

Lists

6. Construct a recursive definition for each of the following functions that involve lists. Use the infix form of cons in the recursive part of each definition. In other words, write $h :: t$ in place of cons(h, t).

   a. $f(n) = \langle 2n, 2(n-1), \ldots, 2, 0\rangle$.
   b. max(L) is the maximum value in nonempty list L of numbers.
   c. $f(x, \langle a_0, \ldots, a_n\rangle) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$.
   d. $f(L)$ is the list of elements x in list L that have property P.
   e. $f(a, \langle x_1, \ldots, x_n\rangle) = \langle x_1 + a, \ldots, x_n + a\rangle$.
   f. $f(a, \langle (x_1, y_1), \ldots, (x_n, y_n)\rangle) = \langle (x_1 + a, y_1), \ldots, (x_n + a, y_n)\rangle$.
   g. $f(n) = \langle (0, n), (1, n-1), \ldots, (n-1, 1), (n, 0)\rangle$. *Hint*: Use part (f).
   h. $f(g, \langle x_1, x_2, \ldots, x_n\rangle) = \langle (x_1, g(x_1)), (x_2, g(x_2)), \ldots, (x_n, g(x_n))\rangle$.
   i. $f(g, h, \langle x_1, \ldots, x_n\rangle) = \langle (g(x_1), h(x_1)), \ldots, (g(x_n), h(x_n))\rangle$.

Using Cat or ConsR

7. Construct a recursive definition for each of the following functions that involve lists. Use the cat operation or consR operation in the recursive part of each definition. (Notice that for any list L and element x we have cat$(L, \langle x\rangle) =$ consR(L, x).)

   a. $f(n) = \langle 0, 1, \ldots, n\rangle$.
   b. $f(n) = \langle 0, 2, 4, \ldots, 2n\rangle$.
   c. $f(n) = \langle 1, 3, 5, \ldots, 2n + 1\rangle$.
   d. $f(n, k) = \langle n, n + 1, n + 2, \ldots, n + \text{k}\rangle$.
   e. $f(n, k) = \langle 0, k, 2k, 3k, \ldots, nk\rangle$.
   f. $f(g, n) = \langle (0, g(0)), (1, g(1)), \ldots, (n , g(n))\rangle$.
   g. $f(n, m) = \langle n, n + 1, n + 2, \ldots, m - 1, m\rangle$, where $n \leq m$.

8. Let *insert* be a function that extends any binary function so that it evalutates a list of two or more arguments. For example,

$$\text{insert}(+, \langle 1, 4, 2, 9\rangle) = 1 + (4 + (2 + 9)) = 16.$$

Write a recursive definition for insert(f, L), where f is any binary function and L is a list of two or more arguments.

9. Write a recursive definition for the function "eq" to check two lists for equality.

10. Write recursive definitions for the following list functions.
    a. The function "last" that returns the last element of a nonempty list. For example, last($\langle a, b, c\rangle$) = c.
    b. The function "front" that returns the list obtained by removing the last element of a nonempty list. For example, front($\langle a, b, c\rangle$) = $\langle a, b\rangle$.

11. Write down a recursive definition for the function "pal" that tests a list of letters to see whether their concatenations form a palindrome. For example, pal($\langle r, a, d, a, r\rangle$) = true since *radar* is a palindrome. *Hint:* Use the functions of Exercise 10.

12. Solve the repeated element problem with the restriction that we want to keep the rightmost occurrence of each repeated element. *Hint:* Use the functions of Exercise 10.

Binary Trees

13. Given the algebraic expression $a + (b\cdot(d + e))$, draw a picture of the binary tree representation of the expression. Then write down the preorder, inorder, and postorder listings of the tree. Are any of the listings familiar to you?

14. Write down recursive definitions for each of the following procedures to print the nodes of a binary tree.
    a. In: Prints the nodes of a binary tree from an inorder traversal.
    b. Post: Prints the nodes of a binary tree from a postorder traversal.

15. Write down recursive definitions for each of the following functions. Include both the equational and if-then-else forms for each definition.
    a. leaves: Returns the number of leaf nodes in a binary tree.
    b. inOrd: Returns the inorder listing of nodes in a binary tree.
    c. postOrd: Returns the postorder listing of nodes in a binary tree.

16. Construct a recursive definition for each of the following functions that involve trees. Represent binary trees as lists where $\langle\ \rangle$ is the empty tree and any nonempty binary tree has the form $\langle L, r, R\rangle$, where r is the root and L and R are its left and right subtrees.
    a. $f(T)$ = sum of values of the nodes of T.
    b. $f(T)$ = depth of a binary tree T. Let the empty tree have depth -1.
    c. $f(T)$ = list of nodes r in binary tree T that have property p.
    d. $f(T)$ = maximum value of nodes in the nonempty binary tree T.

Trees and Algebraic Expressions

17. Recall from Section 1.4 that any algebraic expression can be represented as a tree and the tree can be represented as a list whose head is the root and whose tail is the list of operands in the form of trees. For example, the algebraic expression $a*b + f(c, d, e)$, can be represented by the list

$$\langle +, \langle *, \langle a \rangle, \langle b \rangle \rangle, \langle f, \langle c \rangle, \langle d \rangle, \langle e \rangle \rangle \rangle.$$

   a. Draw the picture of the tree for the given algebraic expression.
   b. Construct a recursive definition for the function "post" that takes an algebraic expression written in the form of a list, as above, and returns the list of nodes in the algebraic expression tree in postfix notation. For example,

$$\text{post}(\langle +, \langle *, \langle a \rangle, \langle b \rangle \rangle, \langle f, \langle c \rangle, \langle d \rangle, \langle e \rangle \rangle \rangle) = \langle a, b, *, c, d, e, f, + \rangle.$$

Relations as Lists of Tuples

18. Construct a recursive definition for each of the following functions that involve lists of tuples. If x is an n-tuple, then x_k represents the kth component of x.
   a. $f(k, L)$ is the list of kth components x_k of tuples x in the list L.
   b. sel(k, a, L) is the list of tuples x in the list L such that $x_k = a$.

Sets Represented as Lists

19. Write a recursive definition for each of the following functions, in which the input arguments are sets represented as lists. Use the primitive operations of cons, head, and tail to build your functions (along with functions already defined):
   a. isMember. For example, isMember$(a, \langle b, a, c \rangle)$ is true.
   b. isSubset. For example, isSubset$(\langle a, b \rangle, \langle b, c, a \rangle)$ is true.
   c. areEqual. For example, areEqual$(\langle a, b \rangle, \langle b, a \rangle)$ is true.
   d. union. For example, union$(\langle a, b \rangle, \langle c, a \rangle) = \langle a, b, c \rangle$.
   e. intersect. For example, intersect$(\langle a, b \rangle, \langle c, a \rangle) = \langle a \rangle$.
   f. difference. For example, difference$(\langle a, b, c \rangle, \langle b, d \rangle) = \langle a, c \rangle$.

Challenges

20. Conway's challenge sequence is defined recursively as follows:

Basis: $f(1) = f(2) = 1$.
Recursion: $f(n) = f(f(n-1)) + f(n - f(n-1))$ for $n > 2$.

Calculate the first 17 elements $f(1), f(2), \ldots, f(17)$. The article by Mallows [1991] contains an account of this sequence.

21. Let fib(k) denote the kth Fibonacci number, and let

$$\text{sum}(k) = 1 + 2 + \cdots + k.$$

Write a recursive definition for the function $f : \mathbb{N} \to \mathbb{N}$ defined by $f(n) = \text{sum}(\text{fib}(n))$. *Hint:* Write down several examples, such as $f(0)$, $f(1)$, $f(2)$, $f(3)$, $f(4)$, … . Then try to find a way to write $f(4)$ in terms of $f(3)$. This might help you discover a pattern.

22. Write a function in if-then-else form to produce the Cartesian product set of two finite sets. You may assume that the sets are represented as lists.

23. We can approximate the square root of a number by using the Newton-Raphson method, which gives an infinite sequence of approximations to the square root of x by starting with an initial guess g. We can define the sequence with the following function:

$$\text{sqrt}(x,\ g) = g :: \text{sqrt}(x,\ (0.5)(g + (x/g))).$$

Find the first three numbers in each of the following infinite sequences, and compare the values with the square root obtained by a calculator.

a. sqrt(4, 1). b. sqrt(4, 2). c. sqrt(4, 3).
d. sqrt(2, 1). e. sqrt(9, 1). f. sqrt(9, 5).

24. Find a definition for each of the following infinite sequence functions.

a. Square: Squares each element in a sequence of numbers.
b. Diff: Finds the difference of the nth and mth numbers of a sequence.
c. Prod: Finds the product of the first n numbers of a sequence.
d. Add: Adds corresponding elements of two numeric sequences.
e. $\text{Skip}(x, k) = \langle x, x + k, x + 2k, x + 3k, \ldots \rangle$.
f. Map: Applies a function to each element of a sequence.
g. ListOf: Finds the list of the first n elements of a sequence.

25. Evaluate each of the following expressions by unfolding the definitions for *Primes* and remove from Example 3.33.

a. head(*Primes*)
b. tail(*Primes*) until reaching the value sieve(remove (2, ints (3))).
c. remove(2, ints (0)) until reaching the value 1 :: 2 :: remove(2, ints (4)).

26. Suppose we define the function $f : \mathbb{N} \to \mathbb{N}$ by

$$f(x) = \text{if } x > 10 \text{ then } x - 10 \text{ else } f(f(x + 11)).$$

This function is recursively defined even though it is not defined by (3.6). Give a simple definition of the function.

3.3 Grammars

Informally, a grammar is a set of rules used to define the structure of the strings in a language. Grammars are important in computer science not only for defining programming languages, but also for defining data sets for programs. Typical applications try to build algorithms that test whether or not an arbitrary string belongs to some language. In this section we'll see that grammars provide a convenient and useful way to describe languages in a fashion similar to an inductive definition, which we discussed in Section 3.1. We'll also see that grammars provide a technique to test whether a string belongs to a language in a fashion similar to the calculation of a recursively defined function, which we described in Section 3.2. So let's get to it.

3.3.1 Recalling English Grammar

We can think of an English sentence as a string of characters if we agree to let the alphabet consist of the usual letters together with the blank character, period, comma, and so on. To *parse* a sentence means break it up into parts that conform to a given grammar.

For example, if an English sentence consists of a subject followed by a predicate, then the sentence

"The big dog chased the cat"

would be broken up into two parts, a subject and a predicate, as follows:

$$\text{subject} = \text{The big dog},$$
$$\text{predicate} = \text{chased the cat}.$$

To denote the fact that a sentence consists of a subject followed by a predicate we'll write the following *grammar rule*:

$$\text{sentence} \to \text{subject predicate}.$$

If we agree that a subject can be an article followed by either a noun or an adjective followed by a noun, then we can break up "The big dog" into smaller parts. The corresponding grammar rule can be written as follows:

$$\text{subject} \to \text{article adjective noun}.$$

Similarly, if we agree that a predicate is a verb followed by an object, then we can break up "chased the cat" into smaller parts. The corresponding grammar rule can be written as follows:

$$\text{predicate} \to \text{verb object}.$$

This is the kind of activity that can be used to detect whether or not a sentence is grammatically correct.

A parsed sentence is often represented as a tree, called the *parse tree* or *derivation tree*. The parse tree for "The big dog chased the cat" is pictured in Figure 3.7.

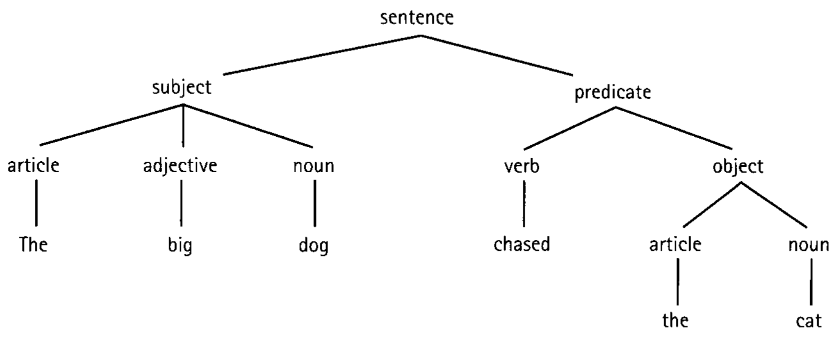


Figure 3.7 Parse tree.

3.3.2 Structure of Grammars

Now that we've recalled a bit of English grammar, let's describe the general structure of grammars for arbitrary languages. If L is a language over an alphabet A, then a grammar for L consists of a set of *grammar rules* of the form

$$\alpha \to \beta,$$

where α and β denote strings of symbols taken from A and from a set of grammar symbols disjoint from A.

The grammar rule $\alpha \to \beta$ is often called a *production*, and it can be read in several different ways as

replace α by β,
α produces β,
α rewrites to β,
α reduces to β.

Every grammar has a special grammar symbol called the *start symbol*, and there must be at least one production with the left side consisting of only the start symbol. For example, if S is the start symbol for a grammar, then there must be at least one production of the form

$$S \to \beta.$$

A Beginning Example

Let's give an example of a grammar for a language and then discuss the process of deriving strings from the productions. Let $A = \{a, b, c\}$. Then a grammar for the language A^* can be described by the following four productions:

$$\begin{aligned} &S \to \Lambda \\ &S \to aS \\ &S \to bS \\ &S \to cS. \end{aligned} \tag{3.11}$$

How do we know that this grammar describes the language A^*? We must be able to describe each string of the language in terms of the grammar rules. For example, let's see how we can use the productions (3.11) to show that the string $aacb$ is in A^*. We'll begin with the start symbol S. Next we'll replace S by the right side of production $S \to aS$. We chose production $S \to aS$ because $aacb$ matches the right hand side of $S \to aS$ by letting $S = acb$. The process of replacing S by aS is called a *derivation*, and we say, "S derives aS." We'll denote this derivation by writing

$$S \Rightarrow aS.$$

The symbol $\Rightarrow$ means "derives in one step." The right-hand side of this derivation contains the symbol S. So we again replace S by aS using the production $S \to aS$ a second time. This results in the derivation

$$S \Rightarrow aS \Rightarrow aaS.$$

The right-hand side of this derivation contains S. In this case we'll replace S by the right side of $S \to cS$. This gives the derivation

$$S \Rightarrow aS \Rightarrow aaS \Rightarrow aacS.$$

Continuing, we replace S by the right side of $S \to bS$. This gives the derivation

$$S \Rightarrow aS \Rightarrow aaS \Rightarrow aacS \Rightarrow aacbS.$$

Since we want this derivation to produce the string $aacb$, we now replace S by the right side of $S \to \Lambda$. This gives the desired derivation of the string $aacb$:

$$S \Rightarrow aS \Rightarrow aaS \Rightarrow aacS \Rightarrow aacbS \Rightarrow aacb\Lambda = aacb.$$

Each step in a derivation corresponds to attaching a new subtree to the parse tree whose root is the start symbol. For example, the parse trees corresponding to the first three steps of our example are shown in Figure 3.8. The completed derivation and parse tree are shown in Figure 3.9.

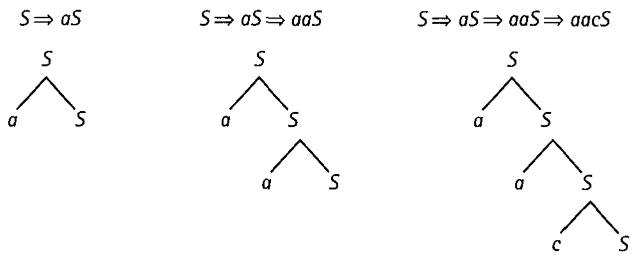

$S \Rightarrow aS$
$S \Rightarrow aS \Rightarrow aaS$
$S \Rightarrow aS \Rightarrow aaS \Rightarrow aacS$

Figure 3.8 Partial derivations and parse trees.

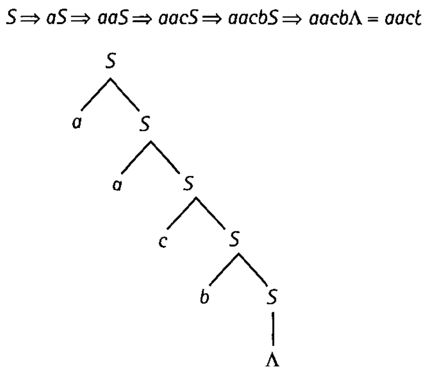

$S \Rightarrow aS \Rightarrow aaS \Rightarrow aacS \Rightarrow aacbS \Rightarrow aacb\Lambda = aacb$

Figure 3.9 Derivation and parse tree.

Definition of a Grammar

Now that we've introduced the idea of a grammar, let's take a minute to describe the four main ingredients of any grammar.

The Four Parts of a Grammar **(3.12)**

1. An alphabet N of grammar symbols called *nonterminals*.
2. An alphabet T of symbols called *terminals*. The terminals are distinct from the nonterminals.
3. A specific nonterminal S, called the *start* symbol.
4. A finite set of productions of the form $\alpha \to \beta$, where α and β are strings over the alphabet $N \cup T$ with the restriction that α is not the empty string. There is at least one production with only the start symbol S on its left side. Each nonterminal must appear on the left side of some production.

Assumption: In this chapter, all grammar productions will have a single nonterminal on the left side. In Chapter 14 we'll see examples of grammars that allow productions to have strings of more than one symbol on the left side.

When two or more productions have the same left side, we can simplify the notation by writing one production with alternate right sides separated by the vertical line |. For example, the four productions (3.11) can be written in the following shorthand form:

$$S \to \Lambda \mid aS \mid bS \mid cS,$$

and we say, "S can be replaced by either Λ , or aS, or bS, or cS."

We can represent a grammar G as a 4-tuple $G = (N, T, S, P)$, where P is the set of productions. For example, if P is the set of productions (3.11), then the grammar can be represented by the 4-tuple

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

The 4-tuple notation is useful for discussing general properties of grammars. But for a particular grammar it's common practice to write down only the productions of the grammar, where the nonterminals are uppercase letters and the first production listed contains the start symbol on its left side. For example, suppose we're given the following grammar:

$$\begin{aligned} S &\to AB \\ A &\to \Lambda \mid aA \\ B &\to \Lambda \mid bB. \end{aligned}$$

We can deduce that the nonterminals are S, A, and B, the start symbol is S, and the terminals are a and b.

3.3.3 Derivations

To discuss grammars further, we need to formalize things a bit. Suppose we're given some grammar. A string made up of terminals and/or nonterminals is called a *sentential form*. Now we can formalize the idea of a derivation.

Definition of Derivation **(3.13)**

If x and y are sentential forms and $\alpha \to \beta$ is a production, then the replacement of α by β in $x\alpha y$ is called a *derivation*, and we denote it by writing

$$x\alpha y \Rightarrow x\beta y.$$

The following three symbols with their associated meanings are used quite often in discussing derivations:

$\Rightarrow$	derives in one step,
$\Rightarrow^+$	derives in one or more steps,
$\Rightarrow^*$	derives in zero or more steps.

For example, suppose we have the following grammar:

$$\begin{aligned} S &\to AB \\ A &\to \Lambda \mid aA \\ B &\to \Lambda \mid bB. \end{aligned}$$

Let's consider the string *aab*. The statement $S \Rightarrow^+ aab$ means that there exists a derivation of *aab* that takes one or more steps. For example, we have

$$S \Rightarrow AB \Rightarrow aAB \Rightarrow aaAB \Rightarrow aaB \Rightarrow aabB \Rightarrow aab.$$

In some grammars it may be possible to find several different derivations of the same string. Two kinds of derivations are worthy of note. A derivation is called a *leftmost derivation* if at each step the leftmost nonterminal of the sentential form is reduced by some production. Similarly, a derivation is called a *rightmost derivation* if at each step the rightmost nonterminal of the sentential form is reduced by some production. For example, the preceding derivation of *aab* is a leftmost derivation. Here's a rightmost derivation of *aab*:

$$S \Rightarrow AB \Rightarrow AbB \Rightarrow Ab \Rightarrow aAb \Rightarrow aaAb \Rightarrow aab.$$

The Language of a Grammar

Sometimes it can be quite difficult, or impossible, to write down a grammar for a given language. So we had better nail down the idea of the language that is associated with a grammar. If G is a grammar, then the *language of G* is the set of terminal strings derived from the start symbol of G. The language of G is denoted by

$$L(G).$$

We can also describe $L(G)$ more formally.

The Language of a Grammar **(3.14)**
If G is a grammar with start symbol S and set of terminals T, then the language of G is the set

$$L(G) = \{s \mid s \in T^* \text{ and } S \Rightarrow^+ s\}.$$

When we're trying to write a grammar for a language, we should at least check to see whether the language is finite or infinite. If the language is finite, then a grammar can consist of all productions of the form $S \to w$ for each string w in the language. For example, the language $\{a, ab\}$ can be described by the grammar $S \to a \mid ab$.

If the language is infinite, then some production or sequence of productions must be used repeatedly to construct the derivations. To see this, notice that there is no bound on the length of strings in an infinite language. Therefore, there is no bound on the number of derivation steps used to derive the strings. If the grammar has n productions, then any derivation consisting of $n + 1$ steps must use some production twice (by the pigeonhole principle).

For example, the infinite language $\{a^n b \mid n \geq 0\}$ can be described by the grammar

$$S \to b \mid aS.$$

To derive the string $a^n b$, we would use the production $S \to aS$ repeatedly—n times to be exact—and then stop the derivation by using the production $S \to b$. The situation is similar to the way we make inductive definitions for sets. For example, the production $S \to aS$ allows us to make the informal statement "If S derives w, then it also derives aw."

Recursive Productions

A production is called *recursive* if its left side occurs on its right side. For example, the production $S \to aS$ is recursive. A production $A \to \alpha$ is *indirectly recursive* if A derives a sentential form that contains A. For example, suppose we have the following grammar:

$$S \to b \mid aA$$
$$A \to c \mid bS.$$

The productions $S \to aA$ and $A \to bS$ are both indirectly recursive because of the following derivations:

$$S \Rightarrow aA \Rightarrow abS,$$
$$A \Rightarrow bS \Rightarrow baA.$$

A grammar is *recursive* if it contains either a recursive production or an indirectly recursive production. So we can make the following more precise statement about grammars for infinite languages:

A grammar for an infinite language must be recursive.

Now let's look at the opposite problem of describing the language of a grammar. We know—by definition—that the language of a grammar is the set of all strings derived from the grammar. But we can also make another interesting observation about any language defined by a grammar:

Any language defined by a grammar is an inductively defined set.

Let's see why this is the case for any grammar G. The following inductive definition does the job, where S denotes the start symbol of G. To simplify the description, we'll say that a derivation is *recursive* if some nonterminal occurs twice due to a recursive production or due to a series of indirectly recursive productions.

Inductive Definition of L(G) **(3.15)**

1. For all strings w that can be derived from S without using a recursive derivation, put w in $L(G)$.
2. If $w \in L(G)$ and there is a derivation of $S \Rightarrow^{+} w$ that contains a nonterminal from a recursive or indirectly recursive production, then use the production to modify the derivation to obtain a new derivation $S \Rightarrow^{+} x$, and put x in $L(G)$.

Proof: Let G be a grammar and let M be the inductive set defined by (3.15). We need to show that $M = L(G)$. It's clear that $M \subset L(G)$ because all strings in M are derived from the start symbol of G. Assume, by way of contradiction, that $M \neq L(G)$. In other words, we have $L(G) - M \neq \varnothing$. Since S derives all the elements of $L(G) - M$, there must be some string $w \in L(G) - M$ that has the shortest leftmost derivation among elements of $L(G) - M$. We can assume that this derivation is recursive. Otherwise, the basis case of (3.15) would force us to put $w \in M$, contrary to our assumption that $w \in L(G) - M$. So the leftmost derivation of w must have the following form, where s and t are terminal strings and α, β, and γ are sentential forms that don't include B:

$$S \Rightarrow^{+} sB\gamma \Rightarrow^{+} stB\beta\gamma \Rightarrow st\alpha\beta\gamma \Rightarrow^{*} w.$$

We can replace $sB\gamma \Rightarrow^{+} stB\beta\gamma$ in this derivation with $sB\gamma \Rightarrow s\alpha\gamma$ to obtain the following derivation of a string u of terminals:

$$S \Rightarrow^{+} sB\gamma \Rightarrow s\alpha\gamma \Rightarrow^{*} u.$$

This derivation is shorter than the derivation of w. So we must conclude that $u \in M$. Now we can apply the induction part of (3.15) to this latter derivation of u to obtain the derivation of w. This tells us that $w \in M$, contrary to our assumption that $w \notin M$. The only thing left for us to conclude is that our assumption that $M \neq L(G)$ was wrong. Therefore $M = L(G)$. QED.

Let's do a simple example to illustrate the use of (3.15).

example 3.34 From Grammar to Inductive Definition

Suppose we're given the following grammar G:

$$S \to \Lambda \mid aB$$
$$B \to b \mid bB.$$

We'll give an inductive definition for $L(G)$. There are two derivations that don't contain recursive productions: $S \Rightarrow \Lambda$ and $S \Rightarrow aB \Rightarrow ab$. This gives us the basis part of the definition for $L(G)$.

Basis: Λ , $ab \in L(G)$.

Now let's find the induction part of the definition. The only recursive production of G is $B \rightarrow bB$. So any element of $L(G)$ whose derivation contains an occurrence of B must have the general form $S \Rightarrow aB \Rightarrow^{+} ay$ for some string y. So we can use the production $B \rightarrow bB$ to add one more step to the derivation as follows:

$$S \Rightarrow aB \Rightarrow abB \Rightarrow^{+} ayy.$$

This gives us the induction step in the definition of $L(G)$.

Induction: If $ay \in L(G)$, then put aby in $L(G)$.

For example, the basis case tells us that $ab \in L(G)$ and the derivation $S \Rightarrow aB \Rightarrow ab$ contains an occurrence of B. So we add one more step to the derivation using the production $B \rightarrow bB$ to obtain the derivation

$$S \Rightarrow aB \Rightarrow abB \Rightarrow abb.$$

So $ab \in L(G)$ implies that $abb \in L(G)$, which in turn implies $ab^3 \in L(G)$, and so on. Thus we can conjecture with some confidence that $L(G)$ is the language $\{\Lambda\} \cup \{ab^n \mid n \in \mathbb{N}\}$.

end example

3.3.4 Constructing Grammars

Now let's get down to business and construct some grammars. We'll start with a few simple examples, and then we'll give some techniques for combining simple grammars. We should note that a language might have more than one grammar. So we shouldn't be surprised when two people come up with two different grammars for the same language.

example 3.35 Three Simple Grammars

We'll write a grammar for each of three simple languages. In each case we'll include a sample derivation of a string in the language. Test each grammar by constructing a few more derivations for strings.

1. $\{\Lambda , a, aa, \ldots, a^n, \ldots\} = \{a^n \mid n \in \mathbb{N}\}$.

   Notice that any string in this language is either Λ or of the form ax for some string x in the language. The following grammar will derive any of these strings:

$$S \rightarrow \Lambda \mid aS.$$

For example, we'll derive the string *aaa*:

$$S \Rightarrow aS \Rightarrow aaS \Rightarrow aaaS \Rightarrow aaa.$$

2. $\{\Lambda\ ,\ ab,\ aabb,\ \ldots,\ a^n b^n,\ \ldots\} = \{a^n b^n \mid n \in \mathbb{N}\}$.

Notice that any string in this language is either Λ or of the form axb for some string x in the language. The following grammar will derive any of these strings:

$$S \rightarrow \Lambda \mid aSb.$$

For example, we'll derive the string *aaabbb*:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb.$$

3. $\{\Lambda\ ,\ ab,\ abab,\ \ldots,\ (ab)^n,\ \ldots\} = \{(ab)^n \mid n \in \mathbb{N}\}$.

Notice that any string in this language is either Λ or of the form abx for some string x in the language. The following grammar will derive any of these strings:

$$S \rightarrow \Lambda \mid abS.$$

For example, we'll derive the string *ababab*:

$$S \Rightarrow abS \Rightarrow ababS \Rightarrow abababS \Rightarrow ababab.$$

end example

Combining Grammars

Sometimes a language can be written in terms of simpler languages, and a grammar can be constructed for the language in terms of the grammars for the simpler languages. We'll concentrate here on the operations of union, product, and closure.

Combining Grammars **(3.16)**

Suppose M and N are languages whose grammars have disjoint sets of nonterminals. (Rename them if necessary.) Suppose also that the start symbols for the grammars of M and N are A and B, respectively. Then we have the following new languages and grammars:

Union Rule: The language $M \cup N$ starts with the two productions

$$S \rightarrow A \mid B.$$

Continued ➡

➡ ➡

Product Rule: The language MN starts with the production

$$S \rightarrow AB.$$

Closure Rule: The language M^* starts with the production

$$S \rightarrow AS \mid \Lambda .$$

example **3.36 Using the Union Rule**

Let's write a grammar for the following language:

$$L = \{\Lambda\ , a, b, aa, bb, \ldots, a^n, b^n, \ldots\}.$$

After some thinking we notice that L can be written as a union $L = M \cup N$, where $M = \{a^n \mid n \in \mathbb{N}\}$ and $N = \{b^n \mid n \in \mathbb{N}\}$. Thus we can write the following grammar for L.

$S \rightarrow A \mid B$	union rule,
$A \rightarrow \Lambda \mid aA$	grammar for M,
$B \rightarrow \Lambda \mid bB$	grammar for N.

end example

example **3.37 Using the Product Rule**

We'll write a grammar for the following language:

$$L = \{a^m b^n \mid m, n \in \mathbb{N}\}.$$

After a little thinking we notice that L can be written as a product $L = MN$, where $M = \{a^m \mid m \in \mathbb{N}\}$ and $N = \{b^n \mid n \in \mathbb{N}\}$. Thus we can write the following grammar for L:

$S \rightarrow AB$	product rule,
$A \rightarrow \Lambda \mid aA$	grammar for M,
$B \rightarrow \Lambda \mid bB$	grammar for N.

end example

example

3.38 Using the Closure Rule

We'll construct a grammar for the language L of all strings with zero or more occurrences of aa or bb. In other words, $L = \{aa, bb\}^*$. If we let $M = \{aa, bb\}$, then $L = M^*$. Thus we can write the following grammar for L.

$$\begin{aligned} &S \to AS \mid \Lambda && \text{closure rule,} \\ &A \to aa \mid bb && \text{grammar for } M. \end{aligned}$$

We can simplify this grammar by substituting for A to obtain the following grammar:

$$S \to aaS \mid bbS \mid \Lambda .$$

end example

example

3.39 Decimal Numerals

We can find a grammar for the language of decimal numerals by observing that a decimal numeral is either a digit or a digit followed by a decimal numeral. The following grammar rules reflect this idea:

$$\begin{aligned} &S \to D \mid DS \\ &D \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9. \end{aligned}$$

We can say that S is replaced by either D or DS, and D can be replaced by any decimal digit. A derivation of the numeral 7801 can be written as follows:

$$S \Rightarrow DS \Rightarrow 7S \Rightarrow 7DS \Rightarrow 7DDS \Rightarrow 78DS \Rightarrow 780S \Rightarrow 780D \Rightarrow 7801.$$

This derivation is not unique. For example, another derivation of 7801 can be written as follows:

$$S \Rightarrow DS \Rightarrow DDS \Rightarrow D8S \Rightarrow D8DS \Rightarrow D80S \Rightarrow D80D \Rightarrow D801 \Rightarrow 7801.$$

end example

example

3.40 Even Decimal Numerals

We can find a grammar for the language of decimal numerals for the even natural numbers by observing that each numeral must have an even digit on its right side. In other words, either it's an even digit or it's a decimal numeral followed by an even digit. The following grammar will do the job:

$$\begin{aligned}
&S \rightarrow E \mid NE\\
&N \rightarrow D \mid DN\\
&E \rightarrow 0 \mid 2 \mid 4 \mid 6 \mid 8\\
&D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9.
\end{aligned}$$

For example, the even numeral 136 has the derivation

$$S \Rightarrow NE \Rightarrow N6 \Rightarrow DN6 \Rightarrow DD6 \Rightarrow D36 \Rightarrow 136.$$

end example

example **3.41 Identifiers**

Most programming languages have identifiers for names of things. Suppose we want to describe a grammar for the set of identifiers that start with a letter of the alphabet followed by zero or more letters or digits. Let S be the start symbol. Then the grammar can be described by the following productions:

$$\begin{aligned}
&S \rightarrow L \mid LA\\
&A \rightarrow LA \mid DA \mid \Lambda\\
&L \rightarrow a \mid b \mid \ldots \mid z\\
&D \rightarrow 0 \mid 1 \mid \ldots \mid 9.
\end{aligned}$$

We'll give a derivation of the string $a2b$ to show that it is an identifier.

$$S \Rightarrow LA \Rightarrow aA \Rightarrow aDA \Rightarrow a2A \Rightarrow a2LA \Rightarrow a2bA \Rightarrow a2b.$$

end example

example **3.42 Some Rational Numerals**

Let's find a grammar for those rational numbers that have a finite decimal representation. In other words, we want to describe a grammar for the language of strings having the form $m.n$ or $-m.n$, where m and n are decimal numerals. For example, 0.0 represents the number 0. Let S be the start symbol. We can start the grammar with the two productions

$$S \rightarrow N.N \mid -N.N.$$

To finish the job, we need to write some productions that allow N to derive a decimal numeral. Try out the following productions:

$$N \rightarrow D \mid DN$$
$$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

end example

example

3.43 Palindromes

We can write a grammar for the set of all palindromes over an alphabet A. Recall that a palindrome is a string that is the same when written in reverse order. For example, let $A = \{a, b, c\}$. Let P be the start symbol. Then the language of palindromes over the alphabet A has the grammar

$$P \rightarrow aPa \mid bPb \mid cPc \mid a \mid b \mid c \mid \Lambda .$$

For example, the palindrome *abcba* can be derived as follows:

$$P \Rightarrow aPa \Rightarrow abPba \Rightarrow abcba.$$

end example

3.3.5 Meaning and Ambiguity

Most of the time we attach meanings to the strings in our lives. For example, the string 3+4 means 7 to most people. The string 3–4–2 may have two distinct meanings to two different people. One person may think that

$$3–4–2 = (3–4)–2 = -3,$$

while another person might think that

$$3–4–2 = 3–(4–2) = 1.$$

If we have a grammar, then we can define the *meaning* of any string in the grammar's language to be the parse tree produced by a derivation. We can often write a grammar so that each string in the grammar's language has exactly one meaning (i.e., one parse tree). When this is not the case, we have an ambiguous grammar. Here's the formal definition.

Definition of Ambiguous Grammar

A grammar is said to be *ambiguous* if its language contains some string that has two different parse trees. This is equivalent to saying that some string has two distinct leftmost derivations or that some string has two distinct rightmost derivations.

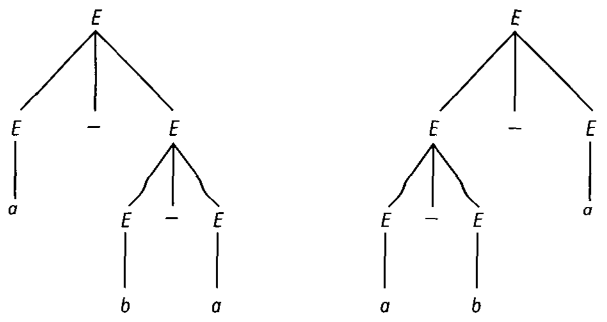


Figure 3.10 Parse trees for an ambiguous string.

To illustrate the ideas, we'll look at some grammars for simple arithmetic expressions. For example, suppose we define a set of arithmetic expressions by the grammar

$$E \rightarrow a \mid b \mid E\text{–}E.$$

The language of this grammar contains strings like a, b, b–a, a–b–a, and b–b–a–b. This grammar is ambiguous because it has a string, namely, a–b–a, that has two distinct parse trees as shown in Figure 3.10.

Since having two distinct parse trees means the same thing as having two distinct leftmost derivations, it's no problem to find the following two distinct leftmost derivations of a–b–a.

$$E \Rightarrow E - E \Rightarrow a - E \Rightarrow a - E - E \Rightarrow a - b - E \Rightarrow a - b - a.$$
$$E \Rightarrow E - E \Rightarrow E - E - E \Rightarrow a - E - E \Rightarrow a - b - E \Rightarrow a - b - a.$$

The two trees in Figure 3.10 reflect the two ways we could choose to evaluate a–b–a. The first tree indicates the meaning

$$a\text{–}b\text{–}a = a\text{–}(b\text{–}a),$$

while the second tree indicates

$$a\text{–}b\text{–}a = (a\text{–}b)\text{–}a.$$

How can we make sure there is only one parse tree for every string in the language? We can try to find a different grammar for the same set of strings. For example, suppose we want a–b–a to mean $(a$–$b)$–a. In other words, we want the first minus sign to be evaluated before the second minus sign. We can give the first minus sign higher precedence than the second by introducing a new nonterminal as shown in the following grammar:

$$E \rightarrow E - T \mid T$$
$$T \rightarrow a \mid b.$$

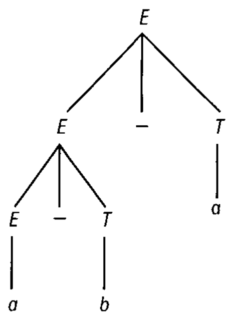


Figure 3.11 Unique parse tree.

Notice that T can be replaced in a derivation only by either a or b. Therefore, every derivation of a–b–a produces the unique parse tree in Figure 3.11.

Exercises

Derivations

1. Given the following grammar.

$$S \rightarrow D \mid DS$$
$$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9.$$

   a. Find the production used in each step of the following derivation.

$$S \Rightarrow DS \Rightarrow 7S \Rightarrow 7DS \Rightarrow 7DDS \Rightarrow 78DS \Rightarrow 780S \Rightarrow 780D \Rightarrow 7801.$$

   b. Find a leftmost derivation of the string 7801.
   c. Find a rightmost derivation of the string 7801.

2. Given the following grammar.

$$S \rightarrow S[S] \mid \Lambda .$$

For each of the following strings, construct a leftmost derivation, a rightmost derivation, and a parse tree.

a. [ ]. b. [[ ]]. c. [ ] [ ]. d. [[ ] [[ ]]].

Constructing Grammars

3. Find a grammar for each of the following languages.

a. $\{bb, bbbb, bbbbbb, \ldots\} = \left\{(bb)^{n+1} \mid n \in \mathbb{N}\right\}$.
b. $\{a, ba, bba, bbba, \ldots\} = \{b^n a \mid n \in \mathbb{N}\}$.
c. $\{\Lambda, ab, abab, ababab, \ldots\} = \{(ab)^n \mid n \in \mathbb{N}\}$.
d. $\{bb, bab, baab, baaab, \ldots\} = \{ba^n b \mid n \in \mathbb{N}\}$.
e. $\left\{ab, abab, \ldots, (ab)^{n+1}, \ldots\right\} = \left\{(ab)^{n+1} \mid n \in \mathbb{N}\right\}$.
f. $\{ab, aabb, \ldots, a^n b^n, \ldots\} = \{a^{n+1} b^{n+1} \mid n \in \mathbb{N}\}$.
g. $\{b, bbb, \ldots, b^{2n+1}, \ldots\} = \{b^{2n+1} \mid n \in \mathbb{N}\}$.
h. $\{b, abc, aabcc, \ldots, a^n bc^n, \ldots\} = \{a^n bc^n \mid n \in \mathbb{N}\}$.
i. $\{ac, abc, abbc, \ldots, ab^n c, \ldots\} = \{ab^n c \mid n \in \mathbb{N}\}$.
j. $\{\Lambda, aa, aaaa, \ldots, a^{2n}, \ldots\} = \{a^{2n} \mid n \in \mathbb{N}\}$.

4. Find a grammar for each language.

   a. $\{a^m b^n \mid m, n \in \mathbb{N}\}$.
   b. $\{a^m bc^n \mid n \in \mathbb{N}\}$.
   c. $\{a^m b^n \mid m, n \in \mathbb{N}$, where $m > 0\}$.
   d. $\{a^m b^n \mid m, n \in \mathbb{N}$, where $n > 0\}$.
   e. $\{a^m b^n \mid m, n \in \mathbb{N}$, where $m > 0$ and $n > 0\}$.

5. Find a grammar for each language.

   a. The even palindromes over $\{a, b, c\}$.
   b. The odd palindromes over $\{a, b, c\}$.
   c. $\{a^{2n} \mid n \in \mathbb{N}\} \cup \{b^{2n+1} \mid n \in \mathbb{N}\}$.
   d. $\{a^n bc^n \mid n \in \mathbb{N}\} \cup \{b^m a^n \mid m, n \in \mathbb{N}\}$
   e. $\{a^m b^n \mid m, n \in \mathbb{N}$, where $m > 0$ or $n > 0\}$.

Mathematical Expressions

6. Find a grammar for each of the following languages.

   a. The set of binary numerals that represent odd natural numbers.
   b. The set of binary numerals that represent even natural numbers.
   c. The set of decimal numerals that represent odd natural numbers.

7. Find a grammar for each of the following languages.

   a. The set of arithmetic expressions that are constructed from decimal numerals, +, and parentheses. Examples: 17, 2+3, (3+(4+5)), and 5+9+20.
   b. The set of arithmetic expressions that are constructed from decimal numerals, − (subtraction), and parentheses, with the property that each expression has only one meaning. For example, 9–34–10 is not allowed.

8. Let the letters a, b, and c be constants; let the letters x, y, and z be variables; and let the letters f and g be functions of arity 1. We can define the set of terms over these symbols by saying that any constant or variable is a term and if t is a term, then so are $f(t)$ and $g(t)$.

   a. Find a grammar for the set of terms.
   b. Find a derivation for the expresssion $f(g(f(x)))$.

9. Let the letters a, b, and c be constants; let the letters x, y, and z be variables; and let the letters f and g be functions of arity 1 and 2, respectively. We can define the set of terms over these symbols by saying that any constant or variable is a term and if s and t are terms, then so are $f(t)$ and $g(s, t)$.

   a. Find a grammar for the set of terms.
   b. Find a derivation for the expresssion $f(g(x, f(b)))$.

10. Find a grammar to capture the precedence $*$ over $+$ in the absence of parentheses. For example, the meaning of $a + b * c$ should be $a + (b * c)$.

Ambiguity

11. Show that each of the following grammars is ambiguous. In other words, find a string that has two different parse trees (equivalently, two different leftmost derivations or two different rightmost derivations).

    a. $S \rightarrow a \mid SbS$.
    b. $S \rightarrow abB \mid AB$ and $A \rightarrow \Lambda \mid Aa$ and $B \rightarrow \Lambda \mid bB$.
    c. $S \rightarrow aS \mid Sa \mid a$.
    d. $S \rightarrow aS \mid Sa \mid b$.
    e. $S \rightarrow S[S]S \mid \Lambda$.
    f. $S \rightarrow Ab \mid A$ and $A \rightarrow b \mid bA$.

Challenges

12. Find a grammar for the language of all strings over $\{a, b\}$ that have the same number of a's and b's.

13. For each grammar, try to find an equivalent grammar that is not ambiguous.

    a. $S \rightarrow a \mid SbS$.
    b. $S \rightarrow abB \mid AB$ and $A \rightarrow \Lambda \mid Aa$ and $B \rightarrow \Lambda \mid bB$.
    c. $S \rightarrow a \mid aS \mid Sa$.
    d. $S \rightarrow b \mid aS \mid Sa$.
    e. $S \rightarrow S[S]S \mid \Lambda$.
    f. $S \rightarrow Ab \mid A$ and $A \rightarrow b \mid bA$.

14. For each grammar, find an equivalent grammar that has no occurrence of Λ on the right side of any rule.

    a. $S \rightarrow AB$
       $A \rightarrow Aa \mid a$
       $B \rightarrow Bb \mid \Lambda$.

    b. $S \rightarrow AcAB$
       $A \rightarrow aA \mid \Lambda$
       $B \rightarrow bB \mid b$.

15. For each grammar G, use (3.15) to find an inductive definition for $L(G)$.

    a. $S \rightarrow \Lambda \mid aaS$.

    b. $S \rightarrow a \mid aBc$ and $b \rightarrow b \mid bB$.

3.4 Chapter Summary

This chapter covered some basic construction techniques that apply to many objects of importance to computer science.

Inductively defined sets are characterized by a basis case, an induction case, and a closure case that is always assumed without comment. The constructors of an inductively defined set are the elements listed in the basis case and the rules specified in the induction case. Many sets of objects used in computer science can be defined inductively—numbers, strings, lists, binary trees, and Cartesian products of sets.

A recursively defined function is defined in terms of itself. Most recursively defined functions have domains that are inductively defined sets. These functions are normally defined by a basis case and a recursive case. The situation is similar for recursively defined procedures. Some infinite sequence functions can be defined recursively. Recursively defined functions and procedures yield powerful programs that are simply stated.

Grammars provide useful ways to describe languages. Grammar productions are used to derive the strings of a language. Any grammar for an infinite language must contain at least one production that is recursive or indirectly recursive. Grammars for different languages can be combined to form new grammars for unions, products, and closures of the languages. Some grammars are ambiguous.

chapter 4

Equivalence, Order, and Inductive Proof

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

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

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

chapter guide

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

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

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

Section 4.4 introduces inductive proof techniques. We'll discuss the technique of mathematical induction for proving statements indexed by the natural numbers. Then we'll extend the discussion to inductive proof techniques for any well-founded set.

4.1 Properties of Binary Relations

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

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

Three Special Properties

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

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

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

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

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

There are two useful opposite properties of the reflexive and symmetric properties.

Two Opposite Properties

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

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

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

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

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

example 4.1 Five Binary Relations

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

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

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

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

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

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

end example

4.1.1 Composition of Relations

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

Definition of Composition

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

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

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

example 4.2 Grandparents

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

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

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

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

end example

example 4.3 Numeric Relations

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

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

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

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

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

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

end example

Other Combining Methods

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

example **4.4 Combining Relations**

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

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

end example

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

Properties of Combining Relation **(4.1)**

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

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

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

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

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

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

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

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

Representations

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

$$R^n.$$

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

$$\begin{aligned} \text{isParentOf}^2 &= \text{isGrandparentOf}, \\ \text{isParentOf}^3 &= \text{isGreatGrandparentOf}. \end{aligned}$$

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

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

example **4.5 Compositions**

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

end example

$$\begin{tikzcd} a \arrow[r] & b \arrow[r] & c \arrow[r] & d \end{tikzcd}$$

R

$$\begin{tikzcd} a \arrow[rr, bend left] & b \arrow[rr, bend right] & c & d \end{tikzcd}$$

R^2

$$\begin{tikzcd} a \arrow[rrr, bend left] & b & c & d \end{tikzcd}$$

R^3

Figure 4.1 Composing a relation.

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

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

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

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

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

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

Inheritance Properties **(4.2)**

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

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

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

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

example 4.6 Integer Relations

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

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

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

$$R^3 = R.$$

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

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

end example

4.1.2 Closures

We've seen how to construct a new relation by composing two existing relations. Let's look at another way to construct a new relation from an existing relation. Here we'll start with a binary relation R and try to construct another relation containing R that also satisfies some particular property. For example, from the "isParentOf" relation, we may want to construct the "isAncestorOf" relation. To discuss this further we need to introduce the idea of closures.

Definition of Closure

If R is a binary relation and p is some property, then the *p closure* of R is the smallest binary relation containing R that satisfies property p. Our goal is to construct closures for each of the three properties reflexive, symmetric, and transitive. We'll denote the *reflexive closure* of R by $r(R)$, the *symmetric closure* of R by $s(R)$, and the *transitive closure* of R by $t(R)$.

To introduce each of the three closures as well as construction techniques, we'll use the following relation on the set $A = \{a, b, c\}$:

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

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

Reflexive Closure

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

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

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

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

Symmetric Closure

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

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

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

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

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

Transitive Closure

To discuss the transitive closure $t(R)$, we'll start with our example. Notice that R contains the pairs (a, b) and (b, c), but (a, c) is not in R. Similarly, R contains the pairs (b, a) and (a, b), but (b, b) is not in R. So $t(R)$ must contain the pairs (a, c) and (b, b). Is there some relation that we can union with R that will add the two needed pairs? The answer is yes, it's R^2. Notice that

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

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

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

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

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

To compute $t(R)$, we need to add the three pairs (a, c), (b, d), and (a, d). In this case, $R^2 = \{(a, c), (b, d)\}$. So the union of R with R^2 is missing (a, d). Can we find another relation to union with R and R^2 that will add this missing pair? Notice that $R^3 = \{(a, d)\}$. So for this example, $t(R)$ is the union

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

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

Constructing the Three Closures

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

Constructing Closures **(4.3)**

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

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

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

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

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

Let's discuss part (4.3d), which assures us that $t(R)$ can be calculated by taking the union of n powers of R if the cardinality of A is n. To see this, notice that any pair $(x, y) \in t(R)$ represents a path from x to y in the digraph of R. Similarly, any pair $(x, y) \in R^k$ represents a path of length k from x to y in the digraph of R. Now if $(x, y) \in R^{n+1}$, then there is a path of length $n + 1$ from x to y in the digraph of R.

Since A has n elements, it follows that some element of A occurs twice in the path from x to y. So there is a shorter path from x to y. Thus $(x, y) \in R^k$ for some $k \leq n$. So nothing new gets added to $t(R)$ by adding powers of R that are higher than the cardinality of A.

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

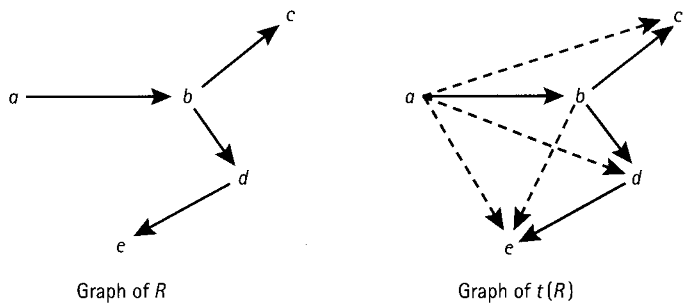


Figure 4.2 R and its transitive closure.

4.7 A Big Transitive Closure

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

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

So the transitive closure of R is the union

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

4.8 A Small Transitive Closure

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

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

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

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

4.9 Generating Less-Than

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

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

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

4.10 Closures of Numeric Relations

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

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

Properties of Closures

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

Inheritance Properties **(4.4)**

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

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

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

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

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

Double Closure Properties **(4.5)**

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

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

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

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

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

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

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

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

4.1.3 Path Problems

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

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

Some Path Problems **(4.6)**

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

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

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

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

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

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

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

Adjacency Matrix

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

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

example **4.11 An Adjacency Matirx**

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

end example

$$\begin{tikzcd} 1 \arrow[r] & 2 \arrow[r] & 3 \arrow[r] & 4 \arrow[l, bend left] \end{tikzcd} \qquad M = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 4.3 Directed graph and adjacency matrix.

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

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

Warshall's Algorithm for Transitive Closure

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

Warshall's Algorithm for Transitive Closure **(4.7)**

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

```
for $k$ := 1 to $n$ do
    for $i$ := 1 to $n$ do
        for $j$ := 1 to $n$ do
            if $(M_{ik} = M_{kj} = 1)$ then $M_{ij}$ := 1
od od od
```

example 4.12 Applying Warshall's Algorithm

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

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

Figure 4.4 Matrix transformations via Warshall's algorithm.

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

end example

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

Floyd's Algorithm for Length of Shortest Path

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

example

4.13 A Weighted Adjacency Matrix

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

end example

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

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

Figure 4.5 Sample weighted adjacency matrix.

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

Floyd's Algorithm for Shortest Distances (4.8)

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

```
for $k := 1$ to $n$ do
    for $i := 1$ to $n$ do
        for $j := 1$ to $n$ do
            $M_{ij} := \min\{M_{ij}, M_{ik} + M_{kj}\}$
od od od
```

example 4.14 Applying Floyd's Algorithm

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

end example

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

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

Figure 4.6 The result of Floyd's algorithm.

Floyd's Algorithm for Finding the Shortest Path

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

Shortest Distances and Shortest Paths Algorithm (4.9)

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

```
for $k := 1$ to $n$ do
    for $i := 1$ to $n$ do
        for $j := 1$ to $n$ do
            if $M_{ik} + M_{kj} < M_{ij}$ then
                $M_{ij} := M_{ik} + M_{kj}$ ;
                $P_{ij} := k$
od od od fi
```

example 4.15 The Path Matrix

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

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

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

Figure 4.7 A path matrix.

sequence 1, 6, 5, and the shortest path between 6 and 4 is the sequence 6, 5, 4. So once we have matrix P from (4.9), it's an easy matter to compute a shortest path between two points. We'll leave this as an exercise.

end example

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

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

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

Exercises

Properties

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

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

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

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

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

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

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

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

Composition

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

   a. isChildOf ∘ isChildOf.

b. isSisterOf ∘ isParentOf.
c. isSonOf ∘ isSiblingOf.
d. isChildOf ∘ isSiblingOf ∘ isParentOf.

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

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

   a. Irreflexive.
   b. Antisymmetric.

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

   a. less ∘ less.
   b. less ∘ less ∘ less.

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

   a. less ∘ greater.
   b. greater ∘ less.
   c. notEqual ∘ less.
   d. greater ∘ notEqual.

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

Closure

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

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

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

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

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

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

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

Path Problems

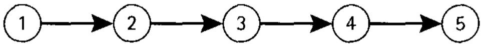
16. Suppose G is the following weighted digraph, where the triple (i, j, d) represents edge (i, j) with distance d:

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

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

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

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



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

Proofs and Challenges

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

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

22. Prove each of the following statements about binary relations.
    a. $R \circ (S \circ T) = (R \circ S) \circ T$. (associativity)
    b. $R \circ (S \cup T) = R \circ S \cup R \circ T$.
    c. $R \circ (S \cap T) \subset R \circ S \cap R \circ T$.

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

24. Prove each of the following statements about a binary relation R over a set A.
    a. If R is reflexive, then $s(R)$ and $t(R)$ are reflexive.
    b. If R is symmetric, then $r(R)$ and $t(R)$ are symmetric.
    c. If R is transitive, then $r(R)$ is transitive.

25. Prove each of the following statements about a binary relation R over a set A.
    a. $rt(R) = tr(R)$.
    b. $rs(R) = sr(R)$.
    c. $st(R) \subset ts(R)$.

4.2 Equivalence Relations

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

The Equality Problem

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

The Equality Problem

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

What is equality? Does it depend on the elements of the set? Why is equality important? What are some properties of equality? We all have an intuitive notion of what equality is because we use it all the time. Equality is important in computer science because programs use equality tests on data. If a programming language doesn't provide an equality test for certain data, then the programmer may need to implement such a test.

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

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

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

$$\text{If } 2+3=1+4, \text{ then } 1+4=2+3.$$
$$\text{If } 2+5=1+6 \text{ and } 1+6=3+4, \text{ then } 2+5=3+4.$$

4.2.1 Definition and Examples

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

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

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

example 4.16 Sample Equivalence Relations

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

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

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

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

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

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

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

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

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

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

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

end example

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

Intersection Property of Equivalence **(4.10)**

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

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

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

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

example 4.17 Equivalent Binary Trees

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

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

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

end example

Equivalence Relations from Functions (Kernel Relations)

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

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

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

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

Kernel Relations **(4.11)**

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

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

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

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

Mod Function Equivalence **(4.12)**

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

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

is an equivalence relation over S.

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

example 4.18 A Numeric Equivalence Relation

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

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

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

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

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

end example

The Equivalence Problem

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

The Equivalence Problem

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

example **4.19 Binary Trees with the Same Structure**

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

```
equiv $(S, T)$ = if $S = \langle\ \rangle$ and $T = \langle\ \rangle$ then true
                 else if $S = \langle\ \rangle$ or $T = \langle\ \rangle$ then false
                 else equiv (left $(S)$, left $(T)$) and equiv (right $(S)$, right $(T)$).
```

end example

4.2.2 Equivalence Classes

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

Equivalence Class

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

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

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

example **4.20 Equivalent Strings**

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

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

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

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

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

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

end example

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

Property of Equivalences **(4.13)**

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

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

4.2.3 Partitions

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

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

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

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

We can do this for any partition of any set.

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

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

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

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

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

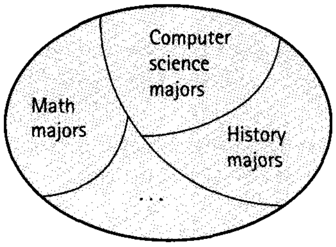


Figure 4.8 A partition of students.

Equivalence Relations and Partitions **(4.14)**

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

For example, suppose we relate two books in the Library of Congress if their call numbers start with the same letter. This relation partitions the set of all the books into 26 subsets, one subset for each letter of the alphabet.

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

example 4.21 Partitioning a Set of Strings

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

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

For each natural number n, the equivalence class $[a^n]$ contains all strings over $\{a, b\}$ that have length n. The partition of S can be written as

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

end example

example **4.22 A Partition of the Natural Numbers**

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

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

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

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

and in general, for any natural number n,

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

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

end example

example **4.23 Partitioning with Mod 5**

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

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

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

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

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

end example

example **4.24 Program Testing**

If the input data set for a program is infinite, then the program can't be tested on every input. However, every program has a finite number of instructions. So we should be able to find a finite data set to cause all instructions of the program to be executed. For example, suppose p is the following program, where x is an integer and q, r, and s represent other parts of the program:

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

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

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

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

end example

Refinement of a Partition

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

For example, the following partitions of $S = \{a, b, c, d\}$ are successive refinements from the coarsest to finest:

$$\begin{array}{ll} \{a, b, c, d\} & \text{(coarsest)} \\ \{a, b\}, \{c, d\} & \\ \{a, b\}, \{c\}, \{d\} & \\ \{a\}, \{b\}, \{c\}, \{d\} & \text{(finest).} \end{array}$$

example 4.25 Partitioning with Mod 2

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

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

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

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

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

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

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

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

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

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

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

end example

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

Intersection Property of Equivalence **(4.15)**

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

example 4.26 Intersecting Equivalence Relations

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

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

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

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

The equivalence classes for F are

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

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

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

end example

example 4.27 Solving the Equality Problem

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

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

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

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

$$1+1+1+1,\ 1+1+11,\ 1+11+1,\ 11+1+1,\ 11+11,\ 1+111,\ 111+1,\ 1111.$$

end example

4.2.4 Generating Equivalence Relations

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

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

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

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

The Smallest Equivalence Relation (4.16)

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

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

example 4.28 Family Trees

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

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

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

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

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

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

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

end example

An Equivalence Problem

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

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

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

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

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

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

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

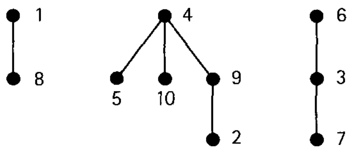


Figure 4.9 Equivalence classes as trees.

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

Figure 4.10 Equivalence classes as an array.

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

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

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

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

Representing Equivalence Classes

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

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

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

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

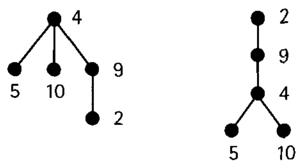


Figure 4.11 Proof that $5 \sim 2$.

$5 \sim 2$, we can reverse the links from 2 to the root of the tree. The before and after pictures are given in Figure 4.11.

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

Kruskal's Algorithm for Minimal Spanning Trees

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

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

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

Kruskal's Algorithm

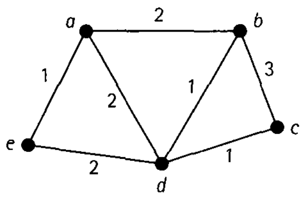
```
1. Sort the edges of the graph by weight, and let $L$ be the sorted list.
2. Let $T$ be the minimal spanning tree and initialize $T := \varnothing$.
3. For each vertex $v$ of the graph, create the equivalence class $[v] = \{v\}$.
4. while there are 2 or more equivalence classes do
           Let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

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

example

4.29 Minimal Spanning Trees

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



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

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

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

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

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

end example

Exercises

Properties

1. Verify that each of the following relations is an equivalence relation.

   a. $x \sim y$ iff x and y are points in a plane equidistant from a fixed point.

   b. $s \sim t$ iff s and t are strings with the same occurrences of each letter.

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

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

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

Equivalence Classes

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

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

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

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

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

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

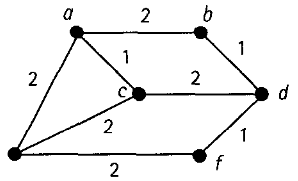
6. Given the following set of words.

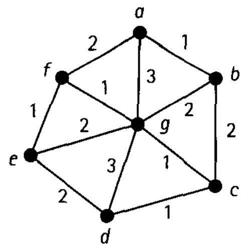
$$\{rot,\ tot,\ root,\ toot,\ roto,\ toto,\ too,\ to,\ otto\}.$$

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

Spanning Trees

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

a. 

b. 

Proofs and Challenges

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

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

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

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

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

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

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

4.3 Order Relations

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

Two Essential Properties of Order

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

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

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

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

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

example 4.30 A Pancake Recipe

Suppose we have the following recipe for making pancakes.

1. Mix the dry ingredients (flour, sugar, baking powder) in a bowl.

2. Mix the wet ingredients (milk, eggs) in a bowl.

3. Mix the wet and dry ingredients together.

4. Oil the pan. (It's an old pan.)

5. Heat the pan.

$$\begin{tikzcd}
1 \arrow[dr] & & & \\
2 \arrow[r] & 3 \arrow[r] & 6 \arrow[r] & 7 \\
4 \arrow[r] & 5 \arrow[ur] & &
\end{tikzcd}$$

Figure 4.12 A pancake recipe.

6. Make a test pancake and throw it away.

7. Make pancakes.

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

$$i \; R \; j \text{ means that step } i \text{ must be done before step } j.$$

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

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

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

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

end example

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

$$i \; R \; j \text{ means that step } i \text{ must be done before step } j.$$

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

4.3.1 Partial Orders

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

Definition of a Partial Order

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

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

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

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

example 4.31 The Divides Relation

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

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

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

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

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

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

It's also easy to see that

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

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

end example

We should note that the literature contains two different definitions of partial order. All definitions require the antisymmetric and transitive properties, but some authors also require the reflexive property. Since we require only the antisymmetric and transitive properties, if a partial order is reflexive and we wish to emphasize it, we'll call it a *reflexive partial order.* For example, $\leq$ is a reflexive partial order on the integers. If a partial order is irreflexive and we wish to emphasize it, we'll call it an *irreflexive partial order.* For example, $<$ is an irreflexive partial order on the integers.

Notation for Partial Orders

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

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

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

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

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

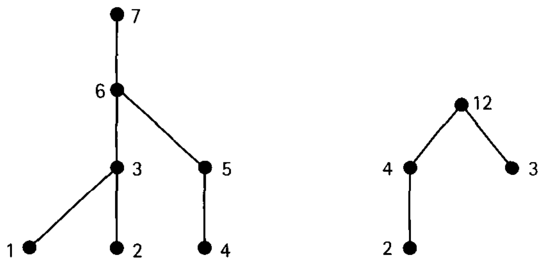
$$\prec \; = \; \preceq - \{(x, x) \mid x \in B\}.$$

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

Chains

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

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



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

Figure 4.13 Two poset diagrams.

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

Predecessors and Successors

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

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

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

Poset Diagrams

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

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

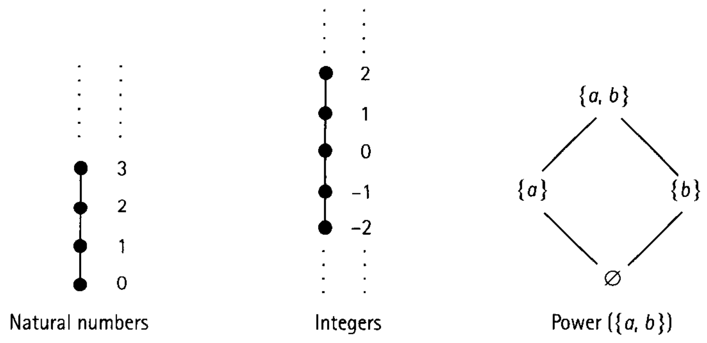


Figure 4.14 Three poset diagrams.

Maxima, Minima, and Bounds

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

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

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

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

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

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

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

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

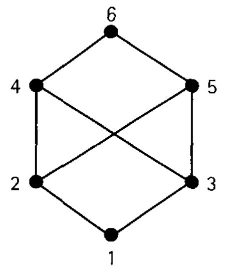


Figure 4.15 A poset diagram.

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

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

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

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

example

4.32 Upper Bounds

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

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

end example

Lattices

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

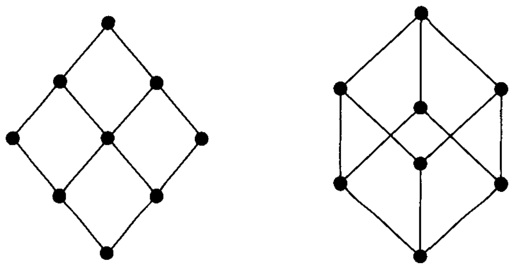


Figure 4.16 Two lattices.

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

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

4.3.2 Topological Sorting

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

The Sorting Problem

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

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

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

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

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

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

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

Topological Sorting

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

The Topological Sorting Problem.

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

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

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

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

Topological Sort Algorithm **(4.17)**

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

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

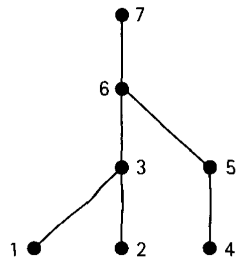


Figure 4.17 Poset of a pancake recipe.

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

Detailed Topological Sort

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

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

example 4.33 A Topological Sort

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

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

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

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

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

You should trace the algorithm for these data representations.

end example

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

4.3.3 Well-Founded Orders

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

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

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

Definition of a Well-Founded Order

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

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

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

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

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

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

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

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

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

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

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

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

The Minimal Element Property

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

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

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

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

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

Notice however that every collection of subsets of a finite set does contain a minimal element. For example, the collection $\{\{a\}, \{b\}, \{a, b\}\}$ has two minimal elements $\{a\}$ and $\{b\}$. Remember, the property that we are looking for must be true for all well-founded sets. So the existence of least elements is out; it's too restrictive.

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

Descending Chains and Minimality **(4.18)**

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

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

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

Lexicographic Ordering of Tuples

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

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

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

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

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

Lexicographic Ordering of Strings

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

Definition

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

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

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

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

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

Standard Ordering of Strings

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

Definition

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

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

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

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

Constructing Well-Founded Orderings

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

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

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

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

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

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

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

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

example 4.34 Some Well-Founded Orderings

a. Any set of lists is well-founded: If L and M are lists, let $L \prec M$ mean length(L) $<$ length(M).

b. Any set of strings is well-founded: If s and t are strings, let $s \prec t$ mean length(s) $<$ length(t).

c. Any set of trees is well-founded: If B and C are trees, let $B \prec C$ mean nodes(B) $<$ nodes(C), where nodes is the function that counts the number of nodes in a tree.

d. Another well-founded ordering on trees can be defined as follows: If B and C are trees, define $B \prec C$ to mean leaves(B) $<$ leaves(C), where leaves is the function that returns the number of leaves in a tree.

e. A well-founded ordering on nonempty trees is defined as follows: For nonempty trees B and C, let $B \prec C$ mean depth(B) $<$ depth(C).

f. The set of all people can be well-founded. Let the age of a person be the floor of the number of years they are old. Then $A \prec B$ if age(A) < age(B). What are the minimal elements?

g. The set $\{\ldots, -3, -2, -1\}$ of negative integers is well-founded: Let $x \prec y$ mean $x > y$.

end example

As the examples show, it's sometimes quite easy to find a well-founded ordering for a set. The next example constructs a finite, hence well-founded, lexicographic order.

example 4.35 A Finite Lexicographic Order

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

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

end example

Inductively Defined Sets are Well-Founded

It's easy to make an inductively defined set W into a well-founded set. We'll give two methods. Both methods let the basis elements of W be the minimal elements of the well-founded order.

Method 1: (4.20)

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

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

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

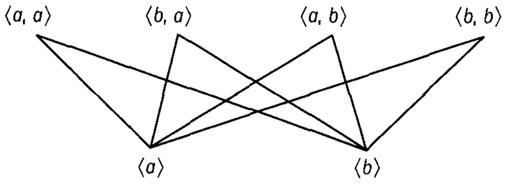


Figure 4.18 Part of a poset diagram.

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

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

If we draw the diagram up to the next level containing triples like $\langle a, a, b \rangle$ and $\langle b, b, a \rangle$, we would have drawn 32 more lines from the two element lists up to the three element lists. So Method 1 relates many elements.

Sometimes it isn't necessary to have an ordering that relates so many elements. This brings us to the second method for defining a well-founded ordering on an inductively defined set W:

Method 2: (4.21)

The ordering $\prec$ is defined as follows:

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

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

The ordering of (4.21) is well-founded because any x can be constructed from basis elements with finitely many constructions. Therefore, there can be

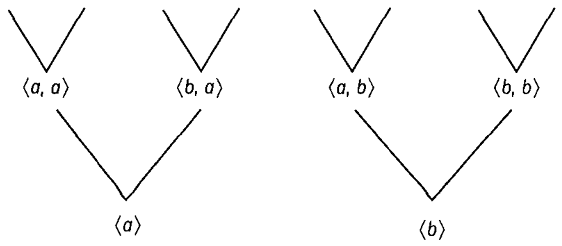


Figure 4.19 Part of a poset diagram.

no infinite descending chain starting at x. With this ordering, there can be many pairs that are not related.

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

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

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

example 4.36 Using One Part of a Product

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

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

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

end example

example 4.37 Using Both Parts of a Product

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

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

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

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

end example

4.3.4 Ordinal Numbers

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

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

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

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

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

the set of natural numbers. The next infinite ordinal is

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

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

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

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

$$0,\ \omega,\ \omega2,\ \omega3, \ldots,\ \omega^2, \ldots,\ \Omega, \ldots.$$

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

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

Exercises

Partial Orders

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

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

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

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

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

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

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

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

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

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

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

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

Well-Founded Property

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

7. For each set S, show that the given partial order on S is well-founded.
   a. Let S be a set of trees. Let $s \prec t$ mean that s has fewer nodes than t.
   b. Let S be a set of trees. Let $s \prec t$ mean that s has fewer leaves than t.
   c. Let S be a set of lists. Let $L \prec M$ mean that length$(L) <$ length(M).

8. Example 4.37 discussed a well-founded ordering for the set $\mathbb{N} \times \mathbb{N}$. Use this ordering to construct two distinct descending chains that start at the pair $(4, 3)$, both of which have maximum length.

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

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

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

Topological Sorting

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

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

Proofs and Challenges

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

13. Prove the two statements of (4.18).

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

4.4 Inductive Proof

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

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

4.4.1 Proof by Mathematical Induction

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

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

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

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

Certainly we don't have the time to check it for the infinity of natural numbers. Is there some other way to prove it? Happily, we will be able to prove the infinitely many cases in just two steps with a technique called *proof by induction,* which we discuss next. If you don't want to see why it works, you can skip ahead to (4.23).

A Basis for Mathematical Induction

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

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

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

A Basis for Mathematical Induction **(4.22)**

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

1. $m \in S$.

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

Then $S = W$.

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

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

The Principle of Mathematical Induction

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

The Principle of Mathematical Induction (4.23)

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

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

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

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

example 4.38 A Correct Formula

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

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

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

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

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

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

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

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

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

end example

example

4.39 A Correct Recusively Defined Function

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

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

Proof: For each $n \in \mathbb{N}$, let $P(n) =$ "$f(n) = 2 + 4 + \cdots + 2n$." We want to show that $P(n)$ is true for all $n \in \mathbb{N}$. To start, notice that $f(0) = 0$. Thus $P(0)$ is true. Now assume that $P(k)$ is true for some $k \in \mathbb{N}$. Now we must furnish a proof that $P(k + 1)$ is true. Starting on the left side of $P(k + 1)$, we have

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

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

end example

A Classic Example: Arithmetic Progressions

When Gauss—mathematician Karl Friedrich Gauss (1777–1855)—was a 10-year-old boy, his schoolmaster, Buttner, gave the class an arithmetic progression of numbers to add up to keep them busy. We should recall that an *arithmetic progression* is a sequence of numbers where each number differs from its successor by the same constant. Gauss wrote down the answer just after Buttner finished writing the problem. Although the formula was known to Buttner, no child of 10 had ever discovered it.

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

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

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

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

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

The Sum of an Arithmetic Progression

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

Sum of Arithmetic Progression **(4.24)**

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

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

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

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

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

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

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

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

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

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

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

Well-Known Sum **(4.25)**

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

A Classic Example: Geometric Progressions

Another important sum adds up the *geometric progression* $1, x, x^2, \ldots, x^n$, where x is any number and n is a natural number. A formula for the sum

$$1 + x + x^2 + \cdots + x^n$$

can be found by multiplying the given expression by the term $x - 1$ to obtain the equation

$$(x-1)\left(1 + x + x^2 + \cdots + x^n\right) = x^{n+1} - 1.$$

Now divide both sides by $x - 1$ to obtain the following formula for the sum of a geometric progression.

$$1 + x + x^2 + \cdots + x^n = \frac{x^{n+1} - 1}{x - 1}. \tag{4.26}$$

The formula works for all $x \neq 1$. We'll prove it by induction.

Proof: If $n = 0$, then both sides are 1. So assume that (4.26) is true for n, and prove that it is true for $n + 1$. Starting with the left-hand side, we have

$$\begin{aligned}1 + x + x^2 + \cdots + x^n + x^{n+1} &= \left(1 + x + x^2 + \cdots + x^n\right) + x^{n+1} \\ &= \frac{x^{n+1} - 1}{x - 1} + x^{n+1} \\ &= \frac{x^{n+1} - 1 + (x - 1)\,x^{n+1}}{x - 1} \\ &= \frac{x^{(n+1)+1} - 1}{x - 1}.\end{aligned}$$

Thus, by (4.23), the formula (4.26) is true for all natural numbers n. QED.

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

4.4.2 Proof by Well-Founded Induction

Let's extend the idea of inductive proof to well-founded sets. Recall that a well-founded set is a poset whose nonempty subsets have minimal elements or, equivalently, every descending chain of elements is finite. We'll start by noticing an easy extension of (4.22) to the case of well-founded sets. If you aren't interested in why the method works, you can skip ahead to (4.28).

The Basis of Well-Founded Induction **(4.27)**

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

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

Then $S = W$.

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

You might notice that condition 1 of (4.27) was not used in the proof. This is because it's a consequence of condition 2 of (4.27). We'll leave this as an exercise (something about an element that doesn't have any predecessors). Condition 1 is stated explicitly because it indicates the first thing that must be done in an inductive proof.

The Technique of Well-Founded Induction

Let's find a more practical form of (4.27) that gives us a technique for proving a collection of statements of the form $P(x)$ for each x in a well-founded set W. The technique is called *well-founded induction.*

Well-Founded Induction **(4.28)**

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

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

Proof: Let $S = \{x \mid x \in W \text{ and } P(x) \text{ is true}\}$. Assume that we have performed the two steps of (4.28). Then S satisfies the hypothesis of (4.27). Therefore $S = W$. In other words, $P(x)$ is true for all $x \in W$. QED.

Second Principle of Mathematical Induction

Now we can state a corollary of (4.28), which lets us make a bigger assumption than we were allowed in (4.23):

Second Principle of Mathematical Induction **(4.29)**

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

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

Proof: Let $W = \{n \mid n \geq m\}$. Notice that W is a well-founded set (actually well-ordered) whose least element is m. Let $S = \{n \mid n \in W \text{ and } P(n) \text{ is true}\}$. Assume that Steps 1 and 2 have been performed. Then $m \in S$, and if $n > m$ and all predecessors of n are in S, then $n \in S$. Therefore, $S = W$, by (4.28). QED.

example **4.40 Products of Primes**

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

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

Proof: For $n \geq 2$, let $P(n)$ be the statement "n is prime or a product of prime numbers." We need to show that $P(n)$ is true for all $n \geq 2$. Since 2 is prime, it follows that $P(2)$ is true. So Step 1 of (4.29) is finished. For Step 2 we'll assume that $n > 2$ and $P(k)$ is true for $2 \leq k < n$. With this assumption we must show that $P(n)$ is true. If n is prime, then $P(n)$ is true. So assume that n is not prime. Then $n = xy$, where $2 \leq x < n$ and $2 \leq y < n$. By our assumption, $P(x)$ and $P(y)$ are both true, which means that x and y are products of primes. Therefore, n is a product of primes. So $P(n)$ is true. Now (4.29) implies that $P(n)$ is true for all $n \geq 2$. QED.

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

end example

Things You Must Do

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

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

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

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

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

4.4.3 A Variety of Examples

Now let's do some examples that do not involve numbers. Thus we'll be using well-founded induction (4.28). We should note that some people refer to well-founded induction as "structural induction" because well-founded sets can contain structures other than numbers, such as lists, strings, binary trees, and Cartesian products of sets. Whatever it's called, let's see how to use it.

example

4.41 Correctness of MakeSet

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

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

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

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

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

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

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

Since $L \prec K$, it follows that $P(L)$ is true. Since isMember(a, L) is false, it follows that the list a :: makeSet(L) has no repeated elements. Thus $P(K)$ is true. Therefore, (4.28) implies that $P(K)$ is true for all lists K. QED.

end example

example

4.42 Using a Lexicographic Ordering

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

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

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

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

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

The result now follows from (4.28). QED.

end example

Inducting on One of Several Variables

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

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

This technique is called *inducting on a single variable.* The form of the statement $P(x, y)$ often gives us a clue as to whether we can induct on a single variable. Here are some examples.

example 4.43 Induction on a Single Variable

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

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

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

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

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

The result now follows from (4.28). QED.

end example

example

4.44 Inserting an Element in a Binary Search Tree

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

```
insert (x, T) = if T = ⟨ ⟩ then tree (⟨ ⟩, x, ⟨ ⟩)
                else if x < root (T) then
                    tree (insert (x, left (T)), root (T), right (T))
                else
                    tree (left (T), root (T), insert (x, right (T))).
```

The claim that we wish to prove is,

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

Proof: We'll induct on the binary tree variable. Our ordering of binary search trees will be based on the number of nodes in a tree. For the basis case we must show that insert(x, $\langle\ \rangle$) is a binary search tree. Since insert(x, $\langle\ \rangle$) = tree($\langle\ \rangle$, x, $\langle\ \rangle$) and a single node tree is a binary search tree, the basis case is true. Next, let T = tree(L, y, R) be a binary search tree, and assume that insert(x, L) and insert(x, R) are binary search trees. Then we must show that insert(x, T) is a binary search tree. There are two cases to consider, depending on whether $x < y$. First, suppose $x < y$. Then we have

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

By the induction assumption it follows that insert(x, L) is a binary search tree. Thus insert(x, T) is a binary search tree. We obtain a similar result if $x \geq y$. It follows from (4.28) that insert(x, T) is a binary search tree for all binary search trees T. QED.

end example

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

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

Proofs about Inductively Defined Sets

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

Closure Property of Inductive Definitions **(4.30)**

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

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

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

example 4.45 Describing an Inductive Set

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

Basis: $1 \in S$.

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

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

Proof: Observe that $1 = 2 \cdot 0 + 1 \in T$ and if $x \in T$, then $x = 2k + 1$ and it follows that $x + 2 = 2(k + 1) + 1 \in T$. So T satisfies the inductive definition. Next we need to show that $T \subset S$. In other words, show that $2k + 1 \in S$ for all $k \in \mathbb{N}$. This calls for an induction proof. If $k = 0$, we have $2 \cdot 0 + 1 = 1 \in S$. Now assume that $2k + 1 \in S$ and show that $2(k + 1) + 1 \in S$. Since $2k + 1 \in S$, the inductive definition tells us that $(2k + 1) + 2 \in S$ and we can write the expression in the form $2(k + 1) + 1 = (2k + 1) + 2 \in S$. Therefore, $2k + 1 \in S$ for all $k \in \mathbb{N}$, which proves that $T \subset S$. So we've proven the three things that allow us to conclude—by the closure property of inductive definitions—that $T = S$. QED.

end example

example 4.46 A Correct Grammar

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

$$S \rightarrow a \mid Sb.$$

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

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

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

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

Now we'll prove that $\{ab^n \mid n \in \mathbb{N}\} = L(G)$. For ease of notation we'll let $M = \{ab^n \mid n \in \mathbb{N}\}$. So we must prove that $M = L(G)$. By (4.30) we must show that M satisfies the basis and induction cases and that $M \subset L(G)$. Then by the closure property of inductive definitions we will infer that $M = L(G)$.

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

tells us that $ab^nb \in L(G)$. But $ab^{n+1} = ab^nb$. So $ab^{n+1} \in L(G)$. Therefore, $M \subset L(G)$. Now the closure property of inductive definitions gives us our conclusion $M = L(G)$. QED.

end example

Exercises

Numbers

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

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

   a. $1 + 3 + 5 + \cdots + (2n - 1) = n^2$.
   b. $5 + 9 + 11 + \cdots + (2n + 3) = n^2 + 4n$.
   c. $3 + 7 + 11 + \cdots + (4n - 1) = n(2n + 1)$.
   d. $2 + 6 + 10 + \cdots + (4n - 2) = 2n^2$.
   e. $1 + 5 + 9 + \cdots + (4n + 1) = (n + 1)(2n + 1)$.
   f. $2 + 8 + 24 + \cdots + n2^n = (n - 1)2^{n+1} + 2$.
   g. $1^2 + 2^2 + \cdots + n^2 = \dfrac{n(n+1)(2n+1)}{6}$.
   h. $2 + 6 + 12 + \cdots + n(n + 1) = \dfrac{n(n+1)(n+2)}{3}$.
   i. $2 + 6 + 12 + \cdots + (n^2 - n) = \dfrac{n(n^2-1)}{3}$.
   j. $(1 + 2 + \cdots + n)^2 = 1^3 + 2^3 + \cdots + n^3$.

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

   a. $F_0 + F_1 + \cdots + F_n = F_{n+2} - 1$.
   b. $F_{n-1}F_{n+1} - F_n^2 = (-1)^n$.
   c. $F_{m+n} = F_{m-1}F_n + F_mF_{n+1}$. *Hint:* Use the lexicographic ordering of $\mathbb{N} \times \mathbb{N}$.
   d. If $m|n$ then $F_m |F_n$. *Hint:* Use the fact that $n = mk$ for some k and show the result by inducting on k with the help of part (c).

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

   a. $L_0 + L_1 + \cdots + L_n = L_{n+2} - 1$.
   b. $L_n = F_{n-1} + F_{n+1}$ for $n \geq 1$, where F_n is the nth Fibonacci number.

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

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

Structures

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

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

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

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

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

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

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

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

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

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

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

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

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

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

a. Assume that the function insert is correct. That is, if S is sorted, then insert(x, S) is also sorted. Prove that sort is correct.

b. Prove that the following definition for insert is correct. That is, prove that insert(x, S) is sorted for all sorted lists S.

$$\begin{aligned}\text{insert}\,(x,S) = {} & \text{if } S = \langle\ \rangle \text{ then } \langle x\rangle \\ & \text{else if } x \le \text{head}\,(S) \text{ then } x :: S \\ & \text{else head}\,(S) :: \text{insert}\,(x, \text{tail}\,(S))\,.\end{aligned}$$

13. Show that the following function g correctly computes the greatest common divisor for each pair of positive integers x and y: *Hint:* (2.2b) might be useful.

$$\begin{aligned}g\,(x,y) = {} & \text{if } x = y \text{ then } x \\ & \text{else if } x > y \text{ then } g\,(x-y,y) \\ & \text{else } g\,(x,y-x)\,.\end{aligned}$$

14. The following program is supposed to input a list of numbers L and output a binary search tree containing the numbers in L:

$$\begin{aligned}f\,(L) = {} & \text{if } L = \langle\ \rangle \text{ then } \langle\ \rangle \\ & \text{else insert}\,(\text{head}\,(L)\,, f\,(\text{tail}\,(L)))\,.\end{aligned}$$

Assume that insert(x, T) correctly returns the binary search tree obtained by inserting the number x in the binary search tree T. Prove the following claim: $f(M)$ is a binary search tree for all lists M.

15. The following program is supposed to return the list obtained by removing the first occurrence of x from the list L.

$$\begin{aligned}\text{delete}\,(x,L) = {} & \text{if } l = \langle\ \rangle \text{ then } \langle\ \rangle \\ & \text{else if } x = \text{head}\,(L) \text{ then tail}\,(L) \\ & \text{else head}\,(L) :: \text{delete}\,(x, \text{tail}\,(L))\,.\end{aligned}$$

Prove that delete performs as expected.

16. The following function claims to remove all occurrences of an element from a list:

$$\begin{aligned}\text{removeAll}\,(a,L) = {} & \text{if } L = \langle\ \rangle \text{ then } L \\ & \text{else if } a = \text{head}\,(L) \text{ then removeAll}\,(a, \text{tail}\,(L)) \\ & \text{else head}\,(L) :: \text{removeAll}\,(a, \text{tail}\,(L))\,.\end{aligned}$$

Prove that removeAll satisfies the claim.

17. Let r stand for the removeAll function from Exercise 16. Prove the following property of r for all elements a, b and all lists L:

$$r(a,\ r(b,\ L)) = r(b,\ r(a,\ L)).$$

18. The following program computes a well-known function called *Ackermann's function*. *Note:* If you try out this function, don't let x and y get too large.

$$\begin{aligned} f(x,y) = {} & \text{if } x = 0 \text{ then } y + 1 \\ & \text{else if } y = 0 \text{ then } f(x-1, 1) \\ & \text{else } f(x-1, f(x, y-1)). \end{aligned}$$

Prove that f is defined for all pairs $(x,\ y)$ in $\mathbb{N} \times \mathbb{N}$. *Hint:* Use the lexicographic ordering on $\mathbb{N} \times \mathbb{N}$. This gives the single basis element $(0,\ 0)$. For the induction assumption, assume that $f(x',\ y')$ is defined for all $(x',\ y')$ such that $(x',\ y') \prec (x,\ y)$. Then show that $f(x,\ y)$ is defined.

19. Let the function "isMember" be defined as follows for any list L:

$$\begin{aligned} \text{isMember}(a, L) = {} & \text{if } L = \langle\ \rangle \text{ then false} \\ & \text{else if } a = \text{head}(L) \text{ then true} \\ & \text{else isMember}(a, \text{tail}(L)). \end{aligned}$$

a. Prove that isMember is correct. That is, show that isMember(a, L) is true if and only if a occurs as an element of L.
b. Prove that the following equation is true for all lists L when $a \neq b$, where removeAll is the function from Exercise 16:

$$\text{isMember}(a,\ \text{removeAll}(b,\ L)) = \text{isMember}(a,\ L).$$

20. Use induction to prove that the following concatenation function is associative.

$$\begin{aligned} \text{cat}(x, y) = {} & \text{if } x = \langle\ \rangle \text{ then } y \\ & \text{else head}(x) :: \text{cat}(\text{tail}(x), y). \end{aligned}$$

In other words, show that cat(x, cat(y, z)) = cat(cat(x, y), z) for all lists x, y, and z.

21. Two students came up with the following two solutions to a problem. Both students used the removeAll function from Exercise 16, which we abbreviate to r.

$$\begin{aligned} \text{Student A:}\quad f(L) &= \text{if } L = \langle\,\rangle \text{ then } \langle\,\rangle \\ &\quad\ \text{else head}(L) :: r(\text{head}(L),\, f(\text{tail}(L))). \\ \text{Student B:}\quad g(L) &= \text{if } L = \langle\,\rangle \text{ then } \langle\,\rangle \\ &\quad\ \text{else head}(L) :: g(r(\text{head}(L),\, \text{tail}(L))). \end{aligned}$$

a. Prove that $r(a, g(L)) = g(r(a, L))$ for all elements a and all lists L. *Hint:* Exercise 17 might be useful in the proof.
b. Prove that $f(L) = g(L)$ for all lists L. *Hint:* Part (a) could be helpful.
c. Can you find an appropriate name for f and g? Can you prove that the name you choose is correct?

Challenges

22. Prove that condition 1 of (4.27) is a consequence of condition 2 of (4.27).

23. Let G be the grammar $S \to a \mid abS$, and let $M = \{(ab)^n a \mid n \in \mathbb{N}\}$. Use (3.12) to construct an inductive definition for $L(G)$. Then use (4.30) to prove that $M = L(G)$.

24. A useful technique for recursively defined functions involves keeping—or accumulating—the results of function calls in *accumulating parameters*: The values in the accumulating parameters can then be used to compute subsequent values of the function that are then used to replace the old values in the accumulating parameters. We call the function by giving initial values to the accumulating parameters. Often these initial values are basis values for an inductively defined set of elements.

    For example, suppose we define the function f as follows:

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

    The second and third arguments to f are accumulating parameters because they always hold two possible values of the function. Prove each of the following statements.

    a. $f(n, 0, 1) = F_n$, the nth Fibonacci number.
    b. $f(n, 2, 1) = L_n$, the nth Lucas number.

    Hint: For part (a), show that $f(n, 0, 1) = f(k, F_{n-k}, F_{n-k+1})$ for $0 \le k \le n$. A similar hint applies to part (b).

25. A *derangement* of a string is an arrangement of the letters of the string such that no letter remains in the same position. In terms of bijections, a derangement of a set S is a bijection f on S such that $f(x) \ne x$ for all x in S. The number of derangements of an n-element set can be given by the

following recursively defined function:

$$\begin{aligned} d(1) &= 0, \\ d(2) &= 1, \\ d(n) &= (n-1)\left(d(n-1)+d(n-2)\right) \quad (n \geq 3). \end{aligned}$$

Give an inductive proof that $d(n) = nd(n-1) + (-1)^n$ for $n \geq 2$.

4.5 Chapter Summary

Binary relations are common denominators for describing the ideas of equivalence, order, and inductive proof. The basic properties that a binary relation may or may not possess are reflexive, symmetric, transitive, irreflexive, and antisymmetric. Binary relations can be constructed from other binary relations by composition and closure, and by the usual set operations. Transitive closure plays an important part in algorithms for solving path problems—Warshall's algorithm, Floyd's algorithm, and the modification of Floyd's algorithm to find shortest paths.

Equivalence relations are characterized by being reflexive, symmetric, and transitive. These relations generalize the idea of basic equality by partitioning a set into classes of equivalent elements. Any set has a hierarchy of partitions ranging from fine to coarse. Equivalence relations can be generated from other relations by taking the transitive symmetric reflexive closure. They can also be generated from functions by the kernel relation. The equivalence problem can be solved by a novel tree structure. Kruskal's algorithm uses an equivalence relation to find a minimal spanning tree for a weighted undirected graph.

Order relations are characterized by being transitive and antisymmetric. Sets with these properties are called posets—for partially ordered sets—because it may be the case that not all pairs of elements are related. The ideas of successor and predecessor apply to posets. Posets can also be topologically sorted. A well-founded poset is characterized by the condition that no descending chain of elements can go on forever. This is equivalent to the condition that any nonempty subset has a minimal element. Well-founded sets can be constructed by mapping objects into a known well-founded set such as the natural numbers. Inductively defined sets are well-founded.

Inductive proof is a powerful technique that can be used to prove infinitely many statements. The most basic inductive proof technique is the principle of mathematical induction. Another useful inductive proof technique is well-founded induction. The important thing to remember about applying inductive proof techniques is to *make an assumption* and then *use the assumption* that you made. Inductive proof techniques can be used to prove properties of recursively defined functions and inductively defined sets.

chapter 5

Analysis Techniques

Remember that time is money.
—Benjamin Franklin (1706–1790)

Time and space are important words in computer science because we want fast algorithms and we want algorithms that don't use a lot of memory. The purpose of this chapter is to study some fundamental techniques and tools that can be used to analyze algorithms for the time and space that they require. Although the study of algorithm analysis is beyond our scope, we'll give some examples to show how the process works.

The cornerstone of algorithm analysis is counting. So after an introduction to the ideas of algorithm analysis, we'll concentrate on techniques to aid the counting process. We'll discuss techniques for finding closed forms for summations. Then we'll discuss permutations, combinations, and discrete probability. Next we'll introduce techniques for solving recurrences. Lastly, with an eye toward comparing algorithms, we'll discuss how to compare the growth rates of functions.

chapter guide

Section 5.1 introduces some ideas about analyzing algorithms. We'll define the worst-case performance of an algorithm and the idea of an optimal algorithm. Then we'll analyze a few example algorithms.

Section 5.2 introduces techniques for finding closed forms for sums that crop up in the analysis of algorithms.

Section 5.3 introduces basic counting techniques for permutations and combinations. We'll also introduce discrete probability so that we can discuss the average-case performance of algorithms.

Section 5.4 introduces techniques for solving recurrences that crop up in the analysis of algorithms.

Section 5.5 introduces techniques for comparing the rates of growth of functions. We'll apply the results to those functions that describe the approximate running time of algorithms.

5.1 Analyzing Algorithms

An important question of computer science is: Can you convince another person that your algorithm is efficient? This takes some discussion. Let's start by stating the following problem.

The Optimal Algorithm Problem

Suppose algorithm A solves problem P. Is A the best solution to P?

What does "best" mean? Two typical meanings are *least time* and *least space.* In either case, we still need to clarify what it means for an algorithm to solve a problem in the least time or the least space. For example, an algorithm running on two different machines may take different amounts of time. Do we have to compare A to every possible solution of P on every type of machine? This is impossible. So we need to make a few assumptions in order to discuss the optimal algorithm problem. We'll concentrate on "least time" as the meaning of "best" because time is the most important factor in most computations.

5.1.1 Worst-Case Running Time

Instead of executing an algorithm on a real machine to find its running time, we'll analyze the algorithm by counting the number of certain operations that it will perform when executed on a real machine. In this way we can compare two algorithms by simply comparing the number of operations of the same type that each performs. If we make a good choice of the type of operations to count, we should get a good measure of an algorithm's performance. For example, we might count addition operations and multiplication operations for a numerical problem. On the other hand, we might choose to count comparison operations for a sorting problem.

The number of operations performed by an algorithm usually depends on the size or structure of the input. The size of the input again depends on the problem. For example, for a sorting problem, "size" usually means the number of items to be sorted. Sometimes inputs of the same size can have different structures that affect the number of operations performed. For example, some sorting algorithms perform very well on an input data set that is all mixed up but perform badly on an input set that is already sorted!

Because of these observations, we need to define the idea of a worst-case input for an algorithm A. An input of size n is a *worst-case input* if, when compared to all other inputs of size n, it causes A to execute the largest number of operations. Now let's get down to business. For any input I we'll denote its size by size(I), and we'll let time(I) denote the number of operations executed by A on I. Then the *worst-case function* for A is defined as follows:

$$W_A(n) = \max\{\text{time}(I) \mid I \text{ is an input and size}(I) = n\}.$$

Now let's discuss comparing different algorithms that solve a problem P. We'll always assume that the algorithms we compare use certain specified operations that we intend to count. If A and B are algorithms that solve P and if $W_A(n) \leq W_B(n)$ for all $n > 0$, then we know algorithm A has worst-case performance that is better than or equal to that of algorithm B. This gives us the proper tool to describe the idea of an optimal algorithm.

Definition of Optimal in the Worst Case
An algorithm A is *optimal in the worst case* for problem P, if for any algorithm B that exists, or ever will exist, the following relationship holds:

$$W_A(n) \leq W_B(n) \text{ for all } n > 0.$$

How in the world can we ever find an algorithm that is optimal in the worst case for a problem P? The answer involves the following three steps:

1. (Find an algorithm) Find or design an algorithm A to solve P. Then do an analysis of A to find the worst-case function W_A.

2. (Find a lower bound) Find a function F such that $F(n) \leq W_B(n)$ for all $n > 0$ and for all algorithms B that solve P.

3. Compare F and W_A. If $F = W_A$, then A is optimal in the worst case.

Suppose we know that $F \neq W_A$ in Step 3. This means that $F(n) < W_A(n)$ for some n. In this case there are two possible courses of action to consider:

1. Put on your construction hat and try to build a new algorithm C such that $W_C(n) \leq W_A(n)$ for all $n > 0$.

2. Put on your analysis hat and try to find a new function G such that $F(n) \leq G(n) \leq W_B(n)$ for all $n > 0$ and for all algorithms B that solve P.

We should note that zero is always a lower bound, but it's not very interesting because most algorithms take more than zero time. A few problems have optimal algorithms. For the vast majority of problems that have solutions, optimal algorithms have not yet been found. The examples contain both kinds of problems.

example

5.1 Matrix Multiplication

We can "multiply" two n by n matrices A and B to obtain the product AB, which is the n by n matrix defined by letting the element in the ith row and jth column of AB be the value of the expression $\sum_{k=1}^{n} A_{ik}B_{kj}$. For example, let A and B be the following 2 by 2 matrices:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}.$$

The product AB is the following 2 by 2 matrix:

$$AB = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}.$$

Notice that the computation of AB takes eight multiplications and four additions. The definition of matrix multiplication of two n by n matrices uses n^3 multiplication operations and $n^2(n - 1)$ addition operations.

A known lower bound for the number of multiplication operations needed to multiply two n by n matrices is n^2. Strassen [1969] showed how to multiply two matrices with about $n^{2.81}$ multiplication operations. The number 2.81 is an approximation to the value of $\log_2(7)$. It stems from the fact that a pair of 2 by 2 matrices can be multiplied by using seven multiplication operations.

Multiplication of larger-size matrices is broken down into multiplying many 2 by 2 matrices. Therefore, the number of multiplication operations becomes less than n^3. This revelation got research going in two camps. One camp is trying to find a better algorithm. The other camp is trying to raise the lower bound above n^2. In recent years, algorithms have been found with still lower numbers. Pan [1978] gave an algorithm to multiply two 70×70 matrices using 143,640 multiplications, which is less than $70^{2.81}$ multiplication operations. Coppersmith and Winograd [1987] gave an algorithm that, for large values of n, uses $n^{2.376}$ multiplication operations. So it goes.

end example

example

5.2 Finding the Minimum

Let's examine an optimal algorithm to find the minimum number in an unsorted list of n numbers. We'll count the number of comparison operations that an algorithm makes between elements of the list. To find the minimum number in a list of n numbers, the minimum number must be compared with the other $n - 1$ numbers. Therefore, $n - 1$ is a lower bound on the number of comparisons needed to find the minimum number in a list of n numbers.

If we represent the list as an array a indexed from 1 to n, then the following algorithm is optimal because the operation $\leq$ is executed exactly $n - 1$ times.

```
$m := a[1]$;
for $i := 2$ to $n$ do
        $m :=$ if $m \leq a[i]$ then $m$ else $a[i]$
od
```

end example

5.1.2 Decision Trees

We can often use a tree to represent the decision processes that take place in an algorithm. A *decision tree* for an algorithm is a tree whose nodes represent decision points in the algorithm and whose leaves represent possible outcomes. Decision trees can be useful in trying to construct an algorithm or trying to find properties of an algorithm. For example, lower bounds may equate to the depth of a decision tree.

If an algorithm makes decisions based on the comparison of two objects, then it can be represented by a *binary decision tree.* Each nonleaf node in the tree represents a pair of objects to be compared by the algorithm, and each branch from that node represents a path taken by the algorithm based on the comparison. Each leaf can represent an outcome of the algorithm. A *ternary decision tree* is similar except that each nonleaf node represents a comparison that has three possible outcomes.

Lower Bounds for Decision Tree Algorithms

Let's see whether we can compute lower bounds for decision tree algorithms. If a decision tree has depth d, then some path from the root to a leaf contains d + 1 nodes. Since the leaf is a possible outcome, it follows that there are d decisions made on the path. Since no other path from the root to a leaf can have more than $d + 1$ nodes, it follows that d is the worst-case number of decisions made by the algorithm.

Now, suppose that a problem has n possible outcomes and it can be solved by a binary decision tree algorithm. What is the best binary decision tree algorithm? We may not know the answer, but we can find a lower bound for the depth of any binary decision tree to solve the problem. Since the problem has n possible outcomes, it follows that any binary decision tree algorithm to solve the problem must have at least n leaves, one for each of the n possible outcomes. Recall that the number of leaves in a binary tree of depth d is at most 2^d.

So if d is the depth of a binary decision tree to solve a problem with n possible outcomes, then we must have $n \leq 2^d$. We can solve this inequality for d by taking $\log_2$ of both sides to obtain $\log_2 n \leq d$. Since d is a natural number, it follows that

$$\lceil \log_2 n \rceil \leq d.$$

In other words, any binary decision tree algorithm to solve a problem with n possible outcomes must have a depth of at least $\lceil \log_2 n \rceil$.

We can do the same analysis for ternary decision trees. The number of leaves in a ternary tree of depth d is at most 3^d. If d is the depth of a ternary decision tree to solve a problem with n possible outcomes, then we must have $n \leq 3^d$. Solve the inequality for d to obtain

$$\lceil \log_3 n \rceil \leq d.$$

In other words, any ternary decision tree algorithm to solve a problem with n possible outcomes must have a depth of at least $\lceil \log_3 n \rceil$.

Many sorting and searching algorithms can be analyzed with decision trees because they perform comparisons. Let's look at some examples to illustrate the idea.

example

5.3 Binary Search

Suppose we search a sorted list in a binary fashion. That is, we check the middle element of the list to see whether it's the key we are looking for. If not, then we perform the same operation on either the left half or the right half of the list, depending on the value of the key. This algorithm has a nice representation as a decision tree. For example, suppose we have the following sorted list of 15 numbers:

$$x_1,\ x_2,\ x_3,\ x_4,\ x_5,\ x_6,\ x_7,\ x_8,\ x_9,\ x_{10},\ x_{11},\ x_{12},\ x_{13},\ x_{14},\ x_{15}.$$

Suppose we're given a number key K, and we must find whether it is in the list. The decision tree for a binary search of the list has the number x_8 at its root. This represents the comparison of K with x_8. If $K = x_8$, then we are successful in one comparison. If $K < x_8$, then we go to the left child of x_8; otherwise we go to the right child of x_8. The result is a ternary decision tree in which the leaves are labeled with either S, for successful search, or U, for unsuccessful search. The decision tree is pictured in Figure 5.1.

Since the depth of the tree is 4, it follows that there will be four comparisons in the worst case to find whether K is in the list. Is this an optimal algorithm? To see that the answer is yes, we can observe that there are 31 possible outcomes for the given problem: 15 leaves labeled with S to represent successful searches; and 16 leaves labeled with U to represent the gaps where $K < x_1$, $x_i < K < x_{i+1}$ for $1 \leq i < 15$, and $x_{15} < K$. Therefore, a worst-case lower bound for the number of comparisons is $\lceil \log_3 31 \rceil = 4$. Therefore the hyphen algorithm is optimal.

end example

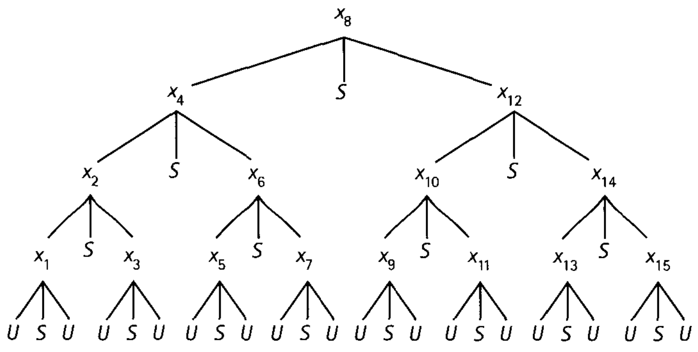


Figure 5.1 Decision tree for binary search.

example

5.4 Finding a Bad Coin

Suppose we are asked to use a pan balance to find the heavy coin among eight coins with the assumption that they all look alike and the other seven all have the same weight. One way to proceed is to always place coins in the two pans that include the bad coin, so the pan will always go down.

This gives a binary decision tree, where each internal node of the tree represents the pan balance. If the left side goes down, then the heavy coin is on the left side of the balance. Otherwise, the heavy coin is on the right side of the balance. Each leaf represents one coin that is the heavy coin. Suppose we label the coins with the numbers 1, 2,..., 8.

The decision tree for one algorithm is shown in Figure 5.2, where the numbers on either side of a nonleaf node represent the coins on either side of the pan balance. This algorithm finds the heavy coin in three weighings. Can we do any better? Look at the next example.

end example

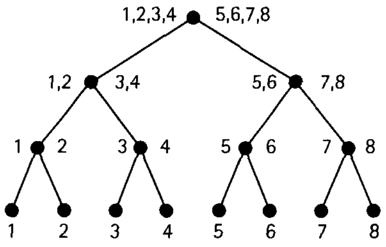


Figure 5.2 A binary decision tree.

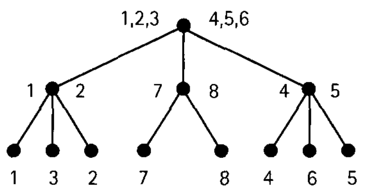


Figure 5.3 An optimal decision tree.

example

5.5 An Optimal Solution

The problem is the same as in Example 5.4. We are asked to use a pan balance to find the heavy coin among eight coins with the assumption that they all look alike and the other seven all have the same weight. In this case we'll weigh coins with the possibility that the two pans are balanced. So a decision tree can have nodes with three children.

We don't have to use all eight coins on the first weighing. For example, Figure 5.3 shows the decision tree for one algorithm. Notice that there is no middle branch on the middle subtree because, at this point, one of the coins 7 or 8 must be the heavy one. This algorithm finds the heavy coin in two weighings.

This algorithm is an optimal pan-balance algorithm for the problem, where we are counting the number of weighings to find the heavy coin. To see this, notice that any one of the eight coins could be the heavy one. Therefore, there must be at least eight leaves on any algorithm's decision tree. But a binary tree of depth d can have 2^d possible leaves. So to get eight leaves, we must have $2^d \geq 8$. This implies that $d \geq 3$. But a ternary tree of depth d can have 3^d possible leaves. So to get eight leaves, we must have $3^d \geq 8$, or $d \geq 2$. Therefore, 2 is a lower bound for the number of weighings. Since the algorithm solves the problem in two weighings, it is optimal.

end example

example

5.6 A Lower Bound Computation

Suppose we have a set of 13 coins in which at most one coin is bad and a bad coin may be heavier or lighter than the other coins. The problem is to use a pan balance to find the bad coin if it exists and say whether it is heavy or light. We'll find a lower bound on the heights of decision trees for pan-balance algorithms to solve the problem.

Any solution must tell whether a bad coin is heavy or light. Thus there are 27 possible conditions: no bad coin and the 13 pairs of conditions (ith coin light, ith coin heavy). Therefore, any decision tree for the problem must have at least 27 leaves. So a ternary decision tree of depth d must satisfy $3^d \geq 27$, or $d \geq 3$. This gives us a lower bound of 3.

Now the big question: Is there an algorithm to solve the problem, where the decision tree of the algorithm has depth 3? The answer is no. Just look at the cases of different initial weighings, and note in each case that the remaining possible conditions cannot be distinguished with just two more weighings. Thus any decision tree for this problem must have depth 4 or more.

end example

Exercises

1. Draw a picture of the decision tree for an optimal algorithm to find the maximum number in the list x_1, x_2, x_3, x_4.

2. Suppose there are 95 possible answers to some problem. For each of the following types of decision tree, find a reasonable lower bound for the number of decisions necessary to solve the problem.

   a. Binary tree. b. Ternary tree. c. Four-way tree.

3. Find a nonzero lower bound on the number of weighings necessary for any ternary pan-balance algorithm to solve the following problem: A set of 30 coins contains at most one bad coin, which may be heavy or light. Is there a bad coin? If so, state whether it's heavy or light.

4. Find an optimal pan-balance algorithm to find a bad coin, if it exists, from 12 coins, where at most one coin is bad (i.e., heavier or lighter than the others). *Hint:* Once you've decided on the coins to weigh for the root of the tree, then the coins that you choose at the second level should be the same coins for all three branches of the tree.

5.2 Finding Closed Forms

In trying to count things we often come up with expressions or relationships that need to be simplified to a form that can be easily computed with familiar operations.

Definition of Closed Form

A *closed form* is an expression that can be computed by applying a fixed number of familiar operations to the arguments. A closed form can't have an ellipsis because the number of operations to evaluate the form would not be fixed. For example, the expression $n(n+1)/2$ is a closed form, but the expression $1 + 2 + \ldots + n$ is not a closed form. In this section we'll introduce some closed forms for sums.

5.2.1 Closed Forms for Sums

Let's start by reviewing a few important facts about summation notation and the indexes used for summing. We can use summation notation to represent a sum like $a_1 + a_2 + \cdots + a_n$ by writing

$$\sum_{i=1}^{n} a_i = a_1 + a_2 + \cdots + a_n.$$

Many problems can be solved by the simple manipulation of sums. So we'll begin by listing some useful facts about sums, which are easily verified.

Summation Facts **(5.1)**

a. $\sum_{i=m}^{n} c = (n - m + 1)c.$

b. $\sum_{i=m}^{n} (a_i + b_i) = \sum_{i=m}^{n} a_i + \sum_{i=m}^{n} b_i.$

c. $\sum_{i=m}^{n} c\, a_i = c \sum_{i=m}^{n} a_i.$

d. $\sum_{i=m}^{n} a_{i+k} = \sum_{i=m+k}^{n+k} a_i.$ (k is any integer)

e. $\sum_{i=m}^{n} a_i x^{i+k} = x^k \sum_{i=m}^{n} a_i x^i.$ (k is any integer)

These facts are very useful in manipulating sums into simpler forms from which we might be able to find closed forms. So we better look at a few closed forms for some elementary sums. We already know some of them.

Closed Forms of Elementary Finite Sums **(5.2)**

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

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

c. $\sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a - 1}$ $(a \neq 1).$

d. $\sum_{i=1}^{n} i a^i = \frac{a - (n+1)a^{n+1} + n a^{n+2}}{(a-1)^2}$ $(a \neq 1).$

These closed forms are quite useful because they pop up in many situations when we are trying to count the number of operations performed by an algorithm. Are closed forms easy to find? Sometimes yes, sometimes no. Many techniques to find closed forms use properties (5.1) to manipulate sums into sums that have known closed forms such as those in (5.2). The following examples show a variety of ways to find closed forms.

example 5.7 A Sum of Odd Numbers

Suppose we need a closed form for the sum of the odd natural numbers up to a certain point:

$$1 + 3 + 5 + \cdots + (2n + 1).$$

We can write this expression using summation notation as follows:

$$\sum_{i=0}^{n} (2i + 1) = 1 + 3 + 5 + \cdots + (2n + 1).$$

Now, let's manipulate the sum to find a closed form. Make sure you can supply the reason for each step:

$$\begin{aligned}\sum_{i=0}^{n} (2i+1) &= \sum_{i=0}^{n} 2i + \sum_{i=0}^{n} 1 \\ &= 2\sum_{i=0}^{n} i + \sum_{i=0}^{n} 1 \\ &= 2\frac{n(n+1)}{2} + (n+1) = (n+1)^2.\end{aligned}$$

end example

example 5.8 Finding a Geometric Progression

Suppose we forgot the closed form for a geometric progression (5.2c). Can we find it again? Here's one way. Let S_n be the following geometric progression, where $a \neq 1$.

$$S_n = 1 + a + a^2 + \cdots + a^n.$$

Notice that we can write the sum S_{n+1} in two different ways in terms of S_n:

$$\begin{aligned}S_{n+1} &= 1 + a + a^2 + \cdots + a^{n+1} \\ &= \left(1 + a + a^2 + \cdots + a^n\right) + a^{n+1} \\ &= S_n + a^{n+1},\end{aligned}$$

and

$$\begin{aligned} S_{n+1} &= 1 + a + a^2 + \cdots + a^{n+1} \\ &= 1 + a\left(1 + a + a^2 + \cdots + a^n\right) \\ &= 1 + aS_n. \end{aligned}$$

So we can equate the two expressions for S_{n+1} to obtain the equation

$$S_n + a^{n+1} = 1 + aS_n.$$

Since $a \neq 1$, we can solve the equation for S_n to obtain the well-known formula (5.2c) for a geometric progression.

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a - 1}.$$

We can verify the answer by mathematical induction on n or by multiplying both sides by $a - 1$ to get an equality.

end example

example **5.9 Finding a Closed Formula**

Let's try to derive the closed formula given in (5.2d) for the sum $\sum_{i=1}^{n} ia^i$. Suppose we let $S_n = \sum_{i=1}^{n} ia^i$. As in Example 5.8, we'll find two expressions for S_{n+1} in terms of S_n. One expression is easy:

$$S_{n+1} = \sum_{i=1}^{n+1} ia^i = \sum_{i=1}^{n} ia^i + (n+1)\,a^{n+1} = S_n + (n+1)\,a^{n+1}.$$

For the other expression we can proceed as follows:

$$\begin{aligned} S_{n+1} &= \sum_{i=1}^{n+1} ia^i \\ &= \sum_{i=0}^{n} (i+1)\,a^{i+1} && \text{(5.1d)} \\ &= \sum_{i=0}^{n} ia^{i+1} + \sum_{i=0}^{n} a^{i+1} && \text{(algebra and 5.1b)} \\ &= a\sum_{i=0}^{n} ia^i + a\sum_{i=0}^{n} a^i && \text{(5.1e)} \\ &= aS_n + a\left(\frac{a^{n+1} - 1}{a - 1}\right) && \text{(5.2c).} \end{aligned}$$

So we can equate the two expressions for S_{n+1} to obtain the equation

$$S_n + (n+1)\,a^{n+1} = aS_n + a\left(\frac{a^{n+1}-1}{a-1}\right).$$

Now solve the equation for S_n to obtain the closed form (5.2d):

$$\sum_{i=1}^{n} ia^i = \frac{a-(n+1)\,a^{n+1}+na^{n+2}}{(a-1)^2}.$$

end example

example **5.10 A Sum of Powers**

A sum like $\sum_{i=1}^{n} i^3$ can be solved in terms of the two sums

$$\sum_{i=1}^{n} i \quad \text{and} \quad \sum_{i=1}^{n} i^2.$$

We'll start by adding the term $(n+1)^4$ to $\sum_{i=1}^{n} i^4$. Then we obtain the following equations.

$$\begin{aligned}\sum_{i=1}^{n} i^4 + (n+1)^4 &= \sum_{i=0}^{n} (i+1)^4 \\ &= \sum_{i=0}^{n} \left(i^4+4i^3+6i^2+4i+1\right) \\ &= \sum_{i=1}^{n} i^4 + 4\sum_{i=1}^{n} i^3 + 6\sum_{i=1}^{n} i^2 + 4\sum_{i=1}^{n} i + \sum_{i=0}^{n} 1.\end{aligned}$$

Now subtract the term $\sum_{i=1}^{n} i^4$ from both sides of the above equation to obtain the following equation:

$$(n+1)^4 = 4\sum_{i=1}^{n} i^3 + 6\sum_{i=1}^{n} i^2 + 4\sum_{i=1}^{n} i + \sum_{i=0}^{n} 1. \qquad (5.3)$$

Since we know the closed forms for the latter three sums on the right side of the equation, we can solve for $\sum_{i=1}^{n} i^3$ to find its closed form. We'll leave this as an exercise. We can use the same method to find a closed form for the expression $\sum_{i=1}^{n} i^k$ for any natural number k.

end example

example

5.11 The Polynomial Problem

Suppose we're interested in the number of arithmetic operations needed to evaluate the following polynomial at some number x.

$$c_0 + c_1x + c_2x^2 + \cdots + c_nx^n.$$

The number of operations performed will depend on how we evaluate it. For example, suppose that we compute each term in isolation and then add up all the terms. There are n addition operations and each term of the form c_ix^i takes i multiplication operations. So the total number of arithmetic operations is given by the following sum:

$$\begin{aligned} n + (0 + 1 + 2 + \ldots + n) &= n + \sum_{i=0}^{n} i \\ &= n + \frac{n + (n+1)}{2} \\ &= \frac{n^2 + 3n}{2}. \end{aligned}$$

So for even small values of n there are many operations to perform. For example, if $n = 30$, then there are 495 arithmetic operations to perform. Can we do better? Sure, we can group terms so that we don't have to repeatedly compute the same powers of x. We'll continue the discussion after we've introduced recurrences.

end example

example

5.12 Simple Sort

In this example we'll construct a simple sorting algorithm and analyze it to find the number of comparison operations. We'll sort an array a of numbers indexed from 1 to n as follows: Find the smallest element in a, and exchange it with the first element. Then find the smallest element in positions 2 through n, and exchange it with the element in position 2. Continue in this manner to obtain a sorted array. To write the algorithm, we'll use a function "min" and a procedure "exchange," which are defined as follows:

min(a, i, n) is the index of the minimum number among the elements $a[i]$, $a[i + 1]$,..., $a[n]$. We can easily modify the algorithm in Example 5.2 on page 276 to accomplish this task with with $n - i$ comparisons.

exchange($a[i]$, $a[j]$) is the usual operation of swapping elements and does not use any comparisons.

Now we can write the sorting algorithm as follows:

```
for $i := 1$ to $n - 1$ do
    $j := \min(a, i, n)$;
    exchange($a[i]$, $a[j]$)
od
```

Now let's compute the number of comparison operations. The algorithm for $\min(a, i, n)$ makes $n - i$ comparisons. So as i moves from 1 to $n - 1$, the number of comparison operations moves from $n - 1$ to $n - (n - 1)$. Adding these comparisons gives the sum of an arithmetic progression,

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

The algorithm makes the same number of comparisons no matter what the form of the input array, even if it is sorted to begin with. So any arrangement of numbers is a worst-case input. For example, to sort 1,000 items it would take 499,500 comparisons, no matter how the items are arranged at the start.

end example

There are many faster sorting algorithms. For example, an algorithm called "heapsort" takes no more than $2n \log_2 n$ comparisons for its worst-case performance. So for 1,000 items, heapsort would take a maximum of 20,000 comparisons—quite an improvement over our simple sort algorithm. In Section 5.3 we'll discover a good lower bound for the worst-case performance of comparison sorting algorithms.

Exercises

Closed Forms for Sums

1. Expand each expression into a sum of terms. Don't evaluate.

   a. $\sum_{i=1}^{5} (2i + 3)$.   b. $\sum_{i=1}^{5} i3^i$.   c. $\sum_{i=0}^{4} (5 - i)\, 3^i$.

2. (*Changing Limits of Summation*). Given the following summation expression:

$$\sum_{i=1}^{n} g(i-1)\, a_i x^{i+1}.$$

   For each of the following lower limits of summation, find an equivalent summation expression that starts with that lower limit.

   a. $i = 0$.   b. $i = 2$.   c. $i = -1$.   d. $i = 3$.   e. $i = -2$.

3. Find a closed form for each of the following sums.

   a. $3 + 6 + 9 + 12 + \cdots + 3n$.
   b. $3 + 9 + 15 + 21 + \cdots + (6n + 3)$.
   c. $3 + 6 + 12 + 24 + \cdots + 3(2^n)$.
   d. $3 + (2)\,3^2 + (3)\,3^3 + (4)\,3^4 + \cdots + n3^n$.

4. For each of the following summations, use summation facts and known closed forms to transform each summation into a closed form.

   a. $\sum_{i=1}^{n} (2i + 2)$.
   b. $\sum_{i=1}^{n} (2i - 1)$.
   c. $\sum_{i=1}^{n} (2i + 3)$.
   d. $\sum_{i=1}^{n} (4i - 1)$.
   e. $\sum_{i=1}^{n} (4i - 2)$.
   f. $\sum_{i=1}^{n} i2^i$.
   g. $\sum_{i=1}^{n} i\,(i + 1)$.
   h. $\sum_{i=2}^{n} \left(i^2 - i\right)$.

5. Solve Equation (5.3) to find a closed form for the expression $\sum_{i=1}^{n} i^3$.

Analyzing Algorithms

6. For the following algorithm, answer each question by giving a formula in terms of n:

```
for $i := 1$ to $n$ do
    for $j := 1$ to $i$ do $x := x + f(x)$ od;
    $x := x + g(x)$
od
```

   a. Find the number of times the assignment statement (:=) is executed during the running of the program.
   b. Find the number of times the addition operation (+) is executed during the running of the program.

7. For the following algorithm, answer each question by giving a formula in terms of n:

```
$i := 1$;
while $i < n + 1$ do
    $i := i + 1$;
    for $j := 1$ to $i$ do $S$ od
od
```

a. Find the number of times the statement S is executed during the running of the program.
b. Find the number of times the assignment statement (:=) is executed during the running of the program.

Proofs and Challenges

8. For the following algorithm, answer each question by giving a formula in terms of n:

```
i := 1;
while i < n + 1 do
    i := i + 2;
    for j := 1 to i do S od
od
```

a. Find the number of times the statement S is executed during the running of the program.
b. Find the number of times the assignment statement (:=) is executed during the running of the program.

5.3 Counting and Discrete Probability

Whenever we need to count the number of ways that some things can be arranged or the number of subsets of things, there are some nice techniques that can help out. That's what our discussion of permutations and combinations will be about.

Whenever we need to count the number of operations performed by an algorithm, we need to consider whether the algorithm yields different results with each execution because of factors such as the size and/or structure of the input data. More generally, many experiments don't always yield the same results each time they are performed. For example, if we flip a coin we can't be sure of the outcome. This brings our discussion to probability. After introducing the basics, we'll see how probability is used when we need to count the number of operations performed by an algorithm in the average case.

5.3.1 Permutations (Order is Important)

In how many different ways can we arrange the elements of a set S? If S has n elements, then there are n choices for the first element. For each of these choices there are $n - 1$ choices for the second element. Continuing in this way, we obtain

$n! = n \cdot (n-1) \cdots 2 \cdot 1$ different arrangements of n elements. Any arrangement of n distinct objects is called a *permutation* of the objects. We'll write down the rule for future use:

Permutations **(5.4)**
There are $n!$ permutations of an n-element set.

For example, if $S = \{a, b, c\}$, then the six possible permutations of S, written as strings, are listed as follows:

$$abc,\ acb,\ bac,\ bca,\ cab,\ cba.$$

Now suppose we want to count the number of permutations of r elements chosen from an n-element set, where $1 \leq r \leq n$. There are n choices for the first element. For each of these choices there are $n - 1$ choices for the second element. We continue this process r times to obtain the answer,

$$n\ (n-1) \cdots (n-r+1).$$

This number is denoted by the symbol $P(n,\ r)$ and is read "the number of permutations of n objects taken r at a time." We should emphasize here that we are counting r distinct objects. So we have the formulas shown in (5.5) and (5.6):

Permutations
The number of permutations of n objects taken r at a time is given by

$$P(n,r) = n(n-1)\cdots(n-r+1), \tag{5.5}$$

which can also be written,

$$P(n,r) = \frac{n!}{(n-r)!}. \tag{5.6}$$

Notice that $P(n,\ 1) = n$ and $P(n,\ n) = n!$. If $S = \{a,\ b,\ c,\ d\}$, then there are 12 permutations of two elements from S, given by the formula $P(4,\ 2) = 4!/2! = 12$. The permutations are listed as follows:

$$ab,\ ba,\ ac,\ ca,\ ad,\ da,\ bc,\ cb,\ bd,\ db,\ cd,\ dc.$$

Permutations with Repeated Elements

Permutations can be thought of as arrangements of objects selected from a set *without replacement.* In other words, we can't pick an element from the set more than once. If we can pick an element more than once, then the objects are said to be selected *with replacement.* In this case the number of arrangements of r objects from an n-element set is just n^r. We can state this idea in terms of bags as follows: The number of distinct permutations of r objects taken from a bag containing n distinct objects, each occurring r times, is n^r. For example, consider the bag $B = [a,\ a,\ b,\ b,\ c,\ c]$. Then the number of distinct permutations of two objects chosen from B is 3^2, and they can be listed as follows:

$$aa,\ ab,\ ac,\ ba,\ bb,\ bc,\ ca,\ cb,\ cc.$$

Let's look now at permutations of all the elements in a bag. For example, suppose we have the bag $B = [a,\ a,\ b,\ b,\ b]$. We can write down the distinct permutations of B as follows:

$$aabbb,\ ababb,\ abbab,\ abbba,\ baabb,\ babab,\ babba,\ bbaab,\ bbaba,\ bbbaa.$$

There are 10 strings. Let's see how to compute the number 10 from the information we have about the bag B. One way to proceed is to place subscripts on the elements in the bag, obtaining the five distinct elements a_1, a_2, b_1, b_2, b_3. Then we get $5! = 120$ permutations of the five distinct elements. Now we remove all the subscripts on the elements, and we find that there are many repeated strings among the original 120 strings.

For example, suppose we remove the subscripts from the two strings,

$$a_1 b_1 b_2 a_2 b_3 \quad \text{and} \quad a_2 b_1 b_3 a_1 b_2\ .$$

Then we obtain two occurrences of the string *abbab*. If we wrote all occurrences down, we would find 12 strings, all of which reduce to the string *abbab* when subscripts are removed. This is because there are 2! permutations of the letters a_1 and a_2, and there are 3! permutations of the letters b_1, b_2, and b_3. So there are $2!3! = 12$ distinct ways to write the string *abbab* when we use subscripts. Of course, the number is the same for any string of two a's and three b's. Therefore, the number of distinct strings of two a's and three b's is found by dividing the total number of subscripted strings by $2!3!$ to obtain $5!/2!3! = 10$. This argument generalizes to obtain the following result about permutations that can contain repeated elements.

Permutations of a Bag **(5.7)**

Let B be an n-element bag with k distinct elements, where each of the numbers $m_1, \ldots, m_k$ denotes the number of occurrences of each element. Then the number of permutations of the n elements of B is

$$\frac{n!}{m_1! \cdots m_k!}.$$

Now let's look at a few examples to see how permutations (5.4)–(5.7) can be used to solve a variety of problems. We'll start with an important result about sorting.

example

5.13 Worst-Case Lower Bound for Comparison Sorting

Let's find a lower bound for the number of comparisons performed by any algorithm that sorts by comparing elements in the list to be sorted. Assume that we have a set of n distinct numbers. Since there are $n!$ possible arrangements of these numbers, it follows that any algorithm to sort a list of n numbers has $n!$ possible input arrangements. Therefore, any decision tree for a comparison sorting algorithm must contain at least $n!$ leaves, one leaf for each possible outcome of sorting one arrangement.

We know that a binary tree of depth d has at most 2^d leaves. So the depth d of the decision tree for any comparison sort of n items must satisfy the inequality

$$n! \le 2^d.$$

We can solve this inequality for the natural number d as follows:

$$\begin{aligned} \log_2 n! &\le d \\ \lceil \log_2 n! \rceil &\le d. \end{aligned}$$

In other words, $\lceil \log_2 n! \rceil$ is a worst-case lower bound for the number of comparisons to sort n items. The number $\lceil \log_2 n! \rceil$ is hard to calculate for large values of n. We'll see in Section 5.5 that it is approximately $n \log_2 n$.

end example

example

5.14 People in a Circle

In how many ways can 20 people be arranged in a circle if we don't count a rotation of the circle as a different arrangement? There are 20! arrangements of 20 people in a line. We can form a circle by joining the two ends of a line. Since there are 20 distinct rotations of the same circle of people, it follows that there are

$$\frac{20!}{20} = 19!$$

distinct arrangements of 20 people in a circle. Another way to proceed is to put one person in a certain fixed position of the circle. Then fill in the remaining 19 people in all possible ways to get 19! arrangements.

end example

example **5.15 Rearranging a String**

How many distinct strings can be made by rearranging the letters of the word *banana*? One letter is repeated twice, one letter is repeated three times, and one letter stands by itself. So we can answer the question by finding the number of permutations of the bag of letters $[b, a, n, a, n, a]$. Therefore, (5.7) gives us the result

$$\frac{6!}{1!2!3!} = 60.$$

end example

example **5.16 Strings with Restrictions**

How many distinct strings of length 10 can be constructed from the two digits 0 and 1 with the restriction that five characters must be 0 and five must be 1? The answer is

$$\frac{10!}{5!5!} = 252$$

because we are looking for the number of permutations from a 10-element bag with five 1's and five 0's.

end example

example **5.17 Constructing a Code**

Suppose we want to build a code to represent each of 29 distinct objects with a binary string having the same minimal length n, where each string has the same number of 0's and 1's. Somehow we need to solve an inequality like

$$\frac{n!}{k!k!} \geq 29,$$

where $k = n/2$. We find by trial and error that $n = 8$. Try it.

end example

5.3.2 Combinations (Order Is Not Important)

Suppose we want to count the number of r-element subsets in an n-element set. For example, if $S = \{a, b, c, d\}$, then there are four 3-element subsets of S: $\{a, b, c\}$, $\{a, b, d\}$, $\{a, c, d\}$, and $\{b, c, d\}$. Is there a formula for the general case? The answer is yes. An easy way to see this is to first count the number of r-element permutations of the n elements, which is given by the formula

$$P(n, r) = \frac{n!}{(n-r)!}.$$

Now each r-element subset has $r!$ distinct r-element permutations, which we have included in our count $P(n, r)$. How do we remove the repeated permutations from the count? Let $C(n, r)$ denote the number of r-element subsets of an n-element set. Since each of the r-element subsets has $r!$ distinct permutations, it follows that $r! \cdot C(n, r) = P(n, r)$. Now divide both sides by $r!$ to obtain the desired formula $C(n, r) = P(n, r)/r!$.

The expression $C(n, r)$ is usually said to represent the number of *combinations* of n things taken r at a time. With combinations, the order in which the objects appear is not important. We count only the different sets of objects. $C(n, r)$ is often read "n choose r." Here's the formula for the record.

Combinations **(5.8)**

The number of *combinations* of n things taken r at a time is given by

$$C(n,r) = \frac{P(n,r)}{r!} = \frac{n!}{r!\,(n-r)!}.$$

example

5.18 Subsets of the Same Size

Let $S = \{a, b, c, d, e\}$. We'll list all the three-element subsets of S:

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

There are 10 such subsets, which we can verify by the formula

$$C(5,3) = \frac{5!}{3!2!} = 10.$$

end example

Binomial Coefficients

Notice how $C(n, r)$ crops up in the following binomial expansion of the expression $(x + y)^4$:

$$\begin{aligned}(x+y)^4 &= x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4\\ &= C(4,0)\,x^4 + C(4,1)\,x^3y + C(4,2)\,x^2y^2 + C(4,3)\,xy^3 + C(4,4)\,y^4.\end{aligned}$$

A useful way to represent $C(n, r)$ is with the *binomial coefficient symbol:*

$$\binom{n}{r} = C(n,r).$$

Using this symbol, we can write the expansion for $(x + y)^4$ as follows:

$$\begin{aligned}(x+y)^4 &= x^4 + 4x^3y + 6x^2y^2 + xy^3 + y^4\\ &= \binom{4}{0}x^4 + \binom{4}{1}x^3y + \binom{4}{2}x^2y^2 + \binom{4}{3}xy^3 + \binom{4}{4}y^4.\end{aligned}$$

This is an instance of a well-known formula called the *binomial theorem*, which can be written as follows, where n is a natural number:

Binomial Theorem **(5.9)**

$$(x+y)^n = \sum_{i=0}^{n} \binom{n}{i} x^{n-i}y^i.$$

Pascal's Triangle

The binomial coefficients for the expansion of $(x + y)^n$ can be read from the nth row of the table in Figure 5.4. The table is called *Pascal's triangle*—after the philosopher and mathematician Blaise Pascal (1623–1662). However, prior to the time of Pascal, the triangle was known in China, India, the Middle East, and Europe. Notice that any interior element is the sum of the two elements above and to its left.

But how do we really know that the following statement is correct?

n	0	1	2	3	4	5	6	7	8	9	10
0	1										
1	1	1									
2	1	2	1								
3	1	3	3	1							
4	1	4	6	4	1						
5	1	5	10	10	5	1					
6	1	6	15	20	15	6	1				
7	1	7	21	35	35	21	7	1			
8	1	8	28	56	70	56	28	8	1		
9	1	9	36	84	126	126	84	36	9	1	
10	1	10	45	120	210	252	210	120	45	10	1

Figure 5.4 Pascal's triangle.

Elements in Pascal's Triangle **(5.10)**

The nth row kth column entry of Pascal's triangle is $\binom{n}{k}$.

Proof: For convenience we will designate a position in the triangle by an ordered pair of the form (row, column). Notice that the edge elements of the triangle are all 1, and they occur at positions $(n, 0)$ or (n, n). Notice also that

$$\binom{n}{0} = 1 = \binom{n}{n}.$$

So (5.10) is true when $k = 0$ or $k = n$. Next, we need to consider the interior elements of the triangle. So let $n > 1$ and $0 < k < n$. We want to show that the element in position (n, k) is $\binom{n}{k}$. To do this, we need the following useful result about binomial coefficients:

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}. \tag{5.11}$$

To prove (5.11), just expand each of the three terms and simplify. Continuing with the proof of (5.10), we'll use well-founded induction. To do this, we need to define a well-founded order on something. For our purposes we will let the something be the set of positions in the triangle. We agree that any position in row $n - 1$ precedes any position in row n. In other words, if $n' < n$, then (n', k') precedes (n, k) for any values of k' and k. Now we can use well-founded induction. We pick position (n, k) and assume that (5.10) is true for all pairs in row $n - 1$. In particular, we can assume that the elements in positions $(n - 1, k)$ and $(n - 1, k - 1)$ have values

$$\binom{n-1}{k} \quad \text{and} \quad \binom{n-1}{k-1}.$$

Now we use this assumption along with (5.11) to tell us that the value of the element in position (n, k) is $\binom{n}{k}$. QED.

Can you find some other interesting patterns in Pascal's triangle? There are lots of them. For example, look down the column labeled 2 and notice that, for each $n \geq 2$, the element in position $(n, 2)$ is the value of the arithmetic sum $1 + 2 + \cdots + (n - 1)$. In other words, we have the formula

$$\binom{n}{2} = \frac{n(n-1)}{2}.$$

Combinations with Repeated Elements

Let's continue our discussion about combinations by counting bags of things rather than sets of things. Suppose we have the set $A = \{a, b, c\}$. How many 3-element bags can we construct from the elements of A? We can list them as follows:

$$[a, a, a], [a, a, b], [a, a, c], [a, b, c], [a, b, b],$$
$$[a, c, c], [b, b, b], [b, b, c], [b, c, c], [c, c, c].$$

So there are ten 3-element bags constructed from the elements of $\{a, b, c\}$.

Let's see if we can find a general formula for the number of k-element bags that can be constructed from an n-element set. For convenience, we'll assume that the n-element set is $A = \{1, 2, \ldots, n\}$. Suppose that $b = [x_1, x_2, x_3, \ldots, x_k]$ is some k-element bag with elements chosen from A, where the elements of b are written so that $x_1 \leq x_2 \leq \cdots \leq x_k$. This allows us to construct the following k-element set:

$$B = \{x_1, x_2 + 1, x_3 + 2, \ldots, x_k + (k - 1)\}.$$

The numbers $x_i + (i - 1)$ are used to ensure that the elements of B are distinct elements in the set $C = \{1, 2, \ldots, n + (k - 1)\}$. So we've associated each k-element bag b over A with a k-element subset B of C. Conversely, suppose that $\{y_1, y_2, y_3, \ldots, y_k\}$ is some k-element subset of C, where the elements are written so that $y_1 \leq y_2 \leq \cdots \leq y_k$. This allows us to construct the k-element bag

$$[y_1, y_2 - 1, y_3 - 2, \ldots, y_k - (k - 1)],$$

whose elements come from the set A. So we've associated each k-element subset of C with a k-element bag over A.

Therefore, the number of k-element bags over an n-element set is exactly the same as the number of k-element subsets of a set with $n + (k - 1)$ elements. This gives us the following result.

Bag Combinations **(5.12)**

The number of k-element bags whose distinct elements are chosen from an n-element set, where k and n are positive, is given by

$$\binom{n+k-1}{k}.$$

example **5.19 Selecting Coins**

In how many ways can four coins be selected from a collection of pennies, nickels, and dimes? Let $S = \{\text{penny, nickel, dime}\}$. Then we need the number of 4-element bags chosen from S. The answer is

$$\binom{3+4-1}{4} = \binom{6}{4} = 15.$$

end example

example **5.20 Selecting a Committee**

In how many ways can five people be selected from a collection of Democrats, Republicans, and Independents? Here we are choosing five-element bags from a set of three characteristics $\{\text{Democrat, Republican, Independent}\}$. The answer is

$$\binom{3+5-1}{5} = \binom{7}{5} = 21.$$

end example

5.3.3 Discrete Probability

The founders of probability theory were Blaise Pascal (1623–1662) and Pierre Fermat (1601–1665). They developed the principles of the subject in 1654 during a correspondence about games of chance. It started when Pascal was asked about a gambling problem. The problem asked how the stakes of a "points" game should be divided up between two players if they quit before either had enough points to win.

Probability comes up whenever we ask about the chance of something happening. To answer such a question requires one to make some kind of assumption. For example, we might ask about the average behavior of an algorithm. That is, instead of the worst-case performance, we might be interested in the average-case performance. This can be a bit tricky because it usually forces us to make one or two assumptions. Some people hate to make assumptions. But it's not so bad. Let's do an example.

Suppose we have a sorted list of the first 15 prime numbers, and we want to know the average number of comparisons needed to find a number in the list, using a binary search. The decision tree for a binary search of the list is pictured in Figure 5.5.

After some thought, you might think it reasonable to add up all the path lengths from the root to a leaf marked with an S (for successful search) and

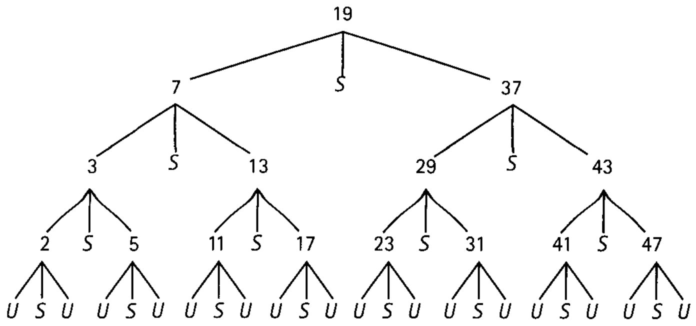


Figure 5.5 Binary search decision tree.

divide by the number of S leaves, which is 15. In this case there are eight paths of length 4, four paths of length 3, two paths of length 2, and one path of length 1. So we get

$$\text{Average path length} = \frac{32+12+4+1}{15} = \frac{49}{15} \approx 3.27.$$

This gives us the average number of comparisons needed to find a number in the list. Or does it? Have we made any assumptions here? Yes, we assumed that each path in the tree has the same chance of being traversed as any other path. Of course, this might not be the case. For example, suppose that we always wanted to look up the number 37. Then the average number of comparisons would be two. So our calculation was made under the assumption that each of the 15 numbers had the same chance of being picked.

Probability Terminology

Let's pause here and introduce some notions and notations for *discrete probability*, which gives us methods to calculate the likelihood of events that have a finite number of outcomes. If some operation or experiment has n possible outcomes and each outcome has the same chance of occurring, then we say that each outcome has *probability* $1/n$. In the preceding example we assumed that each number had probability $1/15$ of being picked. As another example, let's consider the coin-flipping problem. If we flip a fair coin, then there are two possible outcomes, assuming that the coin does not land on its edge. Thus the probability of a head is $1/2$, and the probability of a tail is $1/2$. If we flip the coin 1,000 times, we should expect about 500 heads and 500 tails. So probability has something to do with expectation.

Now for some terminology. The set of all possible outcomes of an experiment is called a *sample space* or *probability space*. The elements in a sample space are called *sample points* or simply *points*. Further, any subset of a sample space is called an *event*. For example, suppose we flip two coins and are interested in the set of possible outcomes. Let H and T mean head and tail, respectively, and let the string HT mean that the first coin lands H and the second coin lands T. Then the sample space for this experiment is the set

$$\{HH,\ HT,\ TH,\ TT\}.$$

For example, the event that one coin lands as a head and the other coin lands as a tail can be represented by the subset $\{HT,\ TH\}$.

To discuss probability we need to make assumptions or observations about the probabilities of sample points. Here is the terminology.

Probability Distribution
A *probability distribution* on a sample space S is an assignment of probabilities to the points of S such that the sum of all the probabilities is 1.

Let's describe a probability distribution from a more formal point of view. Let $S = \{x_1,\ x_2, \ldots,\ x_n\}$ be a sample space. A probability distribution P on S is a function

$$P : S \rightarrow [0,\ 1]$$

such that

$$P(x_1) + P(x_2) + \cdots + P(x_n) = 1.$$

For example, in the two-coin-flip experiment it makes sense to define the following probability distribution on the sample space $S = \{HH,\ HT,\ TH,\ TT\}$:

$$P(HH) = P(HT) = P(TH) = P(TT) = \frac{1}{4}.$$

Probability of an Event

Once we have a probability distribution P defined on the points of a sample space S, we can use P to define the probability of any event E in S.

Probability of an Event
The *probability* of an event E is denoted by $P(E)$ and is defined by

$$P(E) = \sum_{x \in E} P(x).$$

In particular, we have $P(S) = 1$ and $P(\varnothing) = 0$. If A and B are two events, then the following formula follows directly from the definition and the inclusion-exclusion principle.

$$P(A \cup B) = P(A) + P(B) - P(A \cap B).$$

This formula has a very useful consequence. If E' is the complement of E in S, then $S = E \cup E'$ and $E \cap E' = \varnothing$. So it follows from the formula that

$$P(E') = 1 - P(E).$$

example **5.21 Complement of an Event**

In our two-coin-flip example, let E be the event that at least one coin is a tail. Then $E = \{HT, TH, TT\}$. We can calculate $P(E)$ as follows:

$$P(E) = P(\{HT, TH, TT\}) = P(HT) + P(TH) + P(TT) = \tfrac{1}{4} + \tfrac{1}{4} + \tfrac{1}{4} = \tfrac{3}{4}.$$

But we also could observe that the complement E' is the event that both coins are heads. So we could calulate

$$P(E) = 1 - P(E') = 1 - P(HH) = 1 - \tfrac{1}{4} = \tfrac{3}{4}.$$

end example

Classic Example: The Birthday Problem

Suppose we ask 25 people, chosen at random, their birthday (month and day). Would you bet that they all have different birthdays? It seems a likely bet that no two have the same birthday since there are 365 birthdays in the year. But, in fact, the probability that two out of 25 people have the same birthday is greater than 1/2. Again, we're assuming some things here, which we'll get to shortly. Let's see why this is the case. The question we want to ask is:

> Given n people in a room, what is the probability that at least two of the people have the same birthday (month and day)?

We'll neglect leap year and assume that there are 365 days in the year. So there are 365^n possible n-tuples of birthdays for n people. This set of n-tuples is our sample space S. We'll also assume that birthdays are equally distributed throughout the year. So for any n-tuple $(b_1, \ldots, b_n)$ of birthdays, we have $P(b_1, \ldots, b_n) = 1/365^n$. The event E that we are concerned with is the subset of S consisting of all n-tuples that contain two or more equal entries. So our question can be written as follows:

What is $P(E)$?

To answer the question, let's use the complement technique. That is, we'll compute the probability of the event $E' = S - E$, consisting of all n-tuples that have distinct entries. In other words, no two of the n people have the same birthday. Then the probability that we want is $P(E) = 1 - P(E')$. So let's concentrate on E'.

n	$P(E)$
10	0.117
20	0.411
23	0.507
30	0.706
40	0.891

Figure 5.6 Birthday table.

An n-tuple is in E' exactly when all its components are distinct. The cardinality of E' can be found in several ways. For example, there are 365 possible values for the first element of an n-tuple in E'. For each of these 365 values there are 364 values for the second element of an n-tuple in E'. Thus we obtain

$$365 \cdot 364 \cdot 363 \cdots (365 - n + 1)$$

n-tuples in E'. Since each n-tuple of E' is equally likely with probability $1/365^n$, it follows that

$$P(E') = \frac{365 \cdot 364 \cdot 363 \cdots (365 - n + 1)}{365^n}.$$

Thus the probability that we desire is

$$P(E) = 1 - P(E') = 1 - \frac{365 \cdot 364 \cdot 363 \cdots (365 - n + 1)}{365^n}.$$

The table in Figure 5.6 gives a few calculations for different values of n. Notice the case when $n = 23$. The probability is better than 0.5 that two people have the same birthday. Try this out next time you're in a room full of people. It always seems like magic when two people have the same birthday.

example 5.22 Switching Pays

Suppose there is a set of three numbers. One of the three numbers will be chosen as the winner of a three-number lottery. We pick one of the three numbers. Later, we are told that one of the two remaining numbers is not a winner, and we are given the chance to keep the number that we picked or to switch and choose the remaining number. What should we do? We should switch.

To see this, notice that once we pick a number, the probability that we did not pick the winner is 2/3. In other words, it is more likely that one of the other two numbers is a winner. So when we are told that one of the other numbers is not the winner, it follows that the remaining other number has probability 2/3 of being the winner. So go ahead and switch. Try this experiment a few times with a friend to see that in the long run it's better to switch.

Another way to see that switching is the best policy is to modify the problem to a set of 50 numbers and a 50-number lottery. If we pick a number, then the probability that we did not pick a winner is 49/50. Later we are told that 48 of the remaining numbers are not winners, but we are given the chance to keep the number we picked or switch and choose the remaining number. What should we do? We should switch because the chance that the remaining number is the winner is 49/50.

end example

Conditional Probability

If we ask a question about the chance of something happening given that something else has happened, we are using conditional probability.

Conditional Probability
If A and B are events and $P(B) \neq 0$, then *the conditional probability of A given B* is denoted by $P(A|B)$ and defined by

$$P(A|B) = \frac{P(A \cap B)}{P(B)}.$$

We can think of $P(A|B)$ as the probability of the event $A \cap B$ when the sample space is restricted to B once we make an appropriate adjustment to the probability distribution.

example 5.23 Conditional Probability

In a university it is known that 1% of students major in mathematics and 2% major in computer science. Further, it is known that 0.1% of students major in both mathematics and computer science. If a student is chosen at random and happens to be a computer science major, what is the probability that the student is also majoring in mathematics?

To solve the problem we can let A and B be the sets of mathematics majors and computer science majors, respectively. Then the question asks for the value of $P(A|B)$. This is easily calculated because $P(A) = .01$, $P(B) = .02$, and $P(A \cap B) = .001$. Therefore $P(A|B) = .001/.02 = .05$.

end example

Suppose a sample space S is partitioned into disjoint events $E_1, \ldots, E_n$ and B is another event such that $P(B) \neq 0$. Then we can answer some interesting questions about the chance that "B was caused by E_i" for each i. In other words, we can calculate $P(E_i|B)$ for each i. Although this is just a conditional probability, there is an interesting way to compute it in terms of the probabilities $P(B|E_i)$ and $P(E_i)$. The following formula, which is known as *Bayes' theorem*, follows from the assumption that the events $E_1, \ldots, E_n$ form a partition in the sample space.

$$P(E_i|B) = \frac{P(E_i)\,P(B|E_i)}{P(E_1)\,P(B|E_1) + \cdots + P(E_n)\,P(B|E_n)}.$$

When using Bayes' theorem we can think of $P(E_i|B)$ as the probability that B *is caused by* E_i.

example

5.24 Probable Cause

Suppose that the input data set for a program is partitioned into two types, one makes up 60% of the data and the other makes up 40%. Suppose further that inputs from the two types cause warning messages 15% of the time and 20% of the time, respectively. If a random warning message is received, what are the chances that it was caused by an input of each type?

To solve the problem we can use Bayes' theorem. Let E_1 and E_2 be the two sets of data and let B be the set of data that causes warning messages. Then we want to find $P(E_1|B)$ and $P(E_2|B)$. Now we are given the following probabilities:

$$P(E_1) = .6,\ P(E_2) = 0.4,\ P(B|E_1) = .15,\ P(B|E_2) = .2$$

So we can calculate $P(E_1 \mid B)$ as follows:

$$\begin{aligned} P(E_1|B) &= \frac{P(E_1)\,P(B|E_1)}{P(E_1)\,P(B|E_1) + P(E_2)\,P(B|E_2)} \\ &= \frac{(.6)(.15)}{(.6)(.15) + (.4)(.2)} = \frac{.09}{.17} \approx .53. \end{aligned}$$

A similar calculation gives $P(E_2|B) \approx .47$.

end example

Independent Events

Informally, two events A and B are independent if they don't influence each other. If A and B don't influence each other and their probabilities are nonzero, we would like to say that $P(A|B) = P(A)$ and $P(B|A) = P(B)$. This condition will follow from the definition of independence. Two events A and B are *independent* if the following equation holds:

$$P(A \cap B) = P(A) \cdot P(B).$$

It's interesting to note that if A and B are independent events, then so are the three pairs of events A and B', A' and B, and A' and B'. We'll discuss this in the exercises.

The nice thing about independent events is that they simplify the task of assigning probabilities and computing probabilities.

example 5.25 Independence of Events

In the two-coin-flip example, let A be the event that the first coin is heads and let B be the event that the two coins come up different. Then $A = \{HT, HH\}$, $B = \{HT, TH\}$, and $A \cap B = \{HT\}$. If each coin is fair, then A and B are independent because $P(A) = P(B) = 1/2$ and $P(A \cap B) = 1/4$.

Of course many events are not independent. For example, if C is the event that at least one coin is tails, then $C = \{HT, TH, TT\}$. It follows that $A \cap C = \{HT\}$ and $B \cap C = \{HT, TH\}$. If the coins are fair, then it follows that A and C are dependent events and also that B and C are dependent events.

end example

Repeated Independent Trials

Independence is often used to assign probabilities for repeated trials of the same experiment. We'll be content here to discuss repeated trials of an experiment with two outcomes, where the trials are independent. For example, if we flip a coin n times, it's reasonable to assume that each flip is independent of the other flips. To make things a bit more general, we'll assume that a coin comes up either heads with probability p or tails with probability $1 - p$. Here is the question that we want to answer.

What is the probability that the coin comes up heads exactly k times?

To answer this question we need to consider the independence of the flips. For example, if we let A_i be the event that the ith coin comes up heads, then $P(A_i) = p$ and $P(A_i') = 1 - p$. Suppose now that we ask the probability that the first k flips come up heads and the last $n - k$ flips come up tails. Then we are asking about the probability of the event

$$A_1 \cap \cdots \cap A_k \cap A_{k+1}' \cap \cdots \cap A_n'.$$

Since each event in the intersection is independent of the other events, the probability of the intersection is the product of probabilities

$$p^k(1-p)^{n-k}$$

We get the same answer for each arrangement of k heads and $n - k$ tails. So we'll have an answer to the question if we can find the number of different arrangements of k heads and $n - k$ tails. By (5.7) there are

$$\frac{n!}{k!\,(n-k)!}$$

such arrangements. This is also $C(n, k)$, which we can represent by the binomial coefficient symbol. So if a coin flip is repeated n times, then the probability of k successes is given by the expression

$$\binom{n}{k} p^k (1-p)^{n-k}.$$

This set of probabilities is called the *binomial distribution.* The name fits because by the binomial theorem, the sum of the probabilities as k goes from 0 to n is 1. We should note that although we used coin flipping to introduce the ideas, the binomial distribution applies to any experiment with two outcomes that has repeated trials.

Expectation = Average Behavior

Let's get back to talking about averages and expectations. We all know that the average of a bunch of numbers is the sum of the numbers divided by the number of numbers. So what's the big deal? The deal is that we often assign numbers to each outcome in a sample space. For example, in our beginning discussion we assigned a path length to each of the first 15 prime numbers. We added up the 15 path lengths and divided by 15 to get the average. Makes sense, doesn't it? But remember, we assumed that each number was equally likely to occur. This is not always the case. So we also have to consider the probabilities assigned to the points in the sample space.

Let's look at another example to set the stage for a definition of expectation. Suppose we agree to flip a coin. If the coin comes up heads, we agree to pay 4 dollars; if it comes up tails, we agree to accept 5 dollars. Notice here that we have assigned a number to each of the two possible outcomes of this experiment. What is our expected take from this experiment? It depends on the coin. Suppose the coin is fair. After one flip we are either 4 dollars poorer or 5 dollars richer. Suppose we play the game 10 times. What then? Well, since the coin is fair, it seems likely that we can expect to win five times and lose five times. So we can expect to pay 20 dollars and receive 25 dollars. Thus our expectation from 10 flips is 5 dollars.

Suppose we knew that the coin was biased with $P(\text{head}) = 2/5$ and $P(\text{tail}) = 3/5$. What would our expectation be? Again, we can't say much for just one flip. But for 10 flips we can expect about four heads and six tails. Thus we can expect to pay out 16 dollars and receive 30 dollars, for a net profit of 14 dollars. An equation to represent our reasoning follows:

$$10P\,(\text{head})\,(-4) + 10P\,(\text{tail})\,(5) = 10\left(\frac{2}{5}\right)(-4) + 10\left(\frac{3}{5}\right)(5) = \frac{70}{5} = 14.$$

Can we learn anything from this equation? Yes, we can. The 14 dollars represents our take over 10 flips. What's the average profit? Just divide by 10 to get \$1.40. This can be expressed by the following equation:

$$P(\text{head})(-4) + P(\text{tail})(5) = \left(\frac{2}{5}\right)(-4) + \left(\frac{3}{5}\right)(5) = \frac{7}{5} = 1.4.$$

So we can compute the average profit per flip without using the number of coin flips. The average profit per flip is \$1.40 no matter how many flips there are. That's what probability gives us. It's called *expectation*, and we'll generalize from this example to define expectation for any sample space having an assignment of numbers to the sample points.

Definition of Expectation

Let S be a sample space, P a probability distribution on S, and $V : S \to \mathbb{R}$ an assignment of numbers to the points of S. Suppose $S = \{x_1, x_2, \ldots, x_n\}$. Then the *expected value* (or *expectation*) of V is defined by the following formula.

$$E(V) = V(x_1)P(x_1) + V(x_2)P(x_2) + \cdots + V(x_n)P(x_n).$$

So when we want the average behavior, we're really asking for the expectation. For example, in our little coin-flip example we have $S = \{\text{head, tail}\}$, $P(\text{head}) = 2/5$, $P(\text{tail}) = 3/5$, $V(\text{head}) = -4$, and $V(\text{tail}) = 5$. So the expectation of V is calculated by $E(V) = (-4)(2/5) + 5(3/5) = 1.4$.

We should note here that in probability theory the function V is called a *random variable.*

Average Performance of an Algorithm

To compute the average performance of an algorithm A, we must do several things: First, we must decide on a sample space to represent the possible inputs of size n. Suppose our sample space is $S = \{I_1, I_2, \ldots, I_k\}$. Second, we must define a probability distribution P on S that represents our idea of how likely it is that the inputs will occur. Third, we must count the number of operations required by A to process each sample point. We'll denote this count by the function $V : S \to \mathbb{N}$. Lastly, we'll let $\text{Avg}_A(n)$ denote the average number of operations to execute A as a function of input size n. Then $\text{Avg}_A(n)$ is just the expectation of V:

$$\text{Avg}_A(n) = E(V) = V(I_1)P(I_1) + V(I_2)P(I_2) + \cdots + V(I_k)P(I_k).$$

To show that an algorithm A is *optimal in the average case* for some problem, we need to specify a particular sample space and probability distribution. Then we need to show that $\text{Avg}_A(n) \leq \text{Avg}_B(n)$ for all $n > 0$ and for all algorithms B that solve the problem. The problem of finding lower bounds for the average case is just as difficult as finding lower bounds for the worst case. So we're often content to just compare known algorithms to find the best of the bunch.

We'll finish the section with an example showing an average-case analysis of a simple algorithm for sequential search.

Analysis of Sequential Search

Suppose we have the following algorithm to search for an element X in an array L, indexed from 1 to n. If X is in L, the algorithm returns the index of the rightmost occurrence of X. The index 0 is returned if X is not in L:

```
$i := n$;
while $i \geq 1$ and $X \neq L[i]$ do
    $i := i - 1$
od
```

We'll count the average number of comparisons $X \neq L[i]$ performed by the algorithm. Frst we need a sample space. Suppose we let I_i denote the input case where the rightmost occurrence of X is at the ith position of L. Let I_{n+1} denote the case in which X is not in L. So the sample space is the set

$$\{I_1, I_2, \ldots, I_{n+1}\}.$$

Let $V(I)$ denote the number of comparisons made by the algorithm when the input has the form I. Looking at the algorithm, we obtain

$$\begin{aligned} V(I_i) &= n - i + 1 \quad \text{for } 1 \leq i \leq n, \\ V(I_{n+1}) &= n. \end{aligned}$$

Suppose we let q be the probability that X is in L. Thus $1 - q$ is the probability that X is not in L. Let's also assume that whenever X is in L, its position is random. This gives us the following probability distribution P over the sample space:

$$\begin{aligned} P(I_i) &= \frac{q}{n} \quad \text{for } 1 \leq i \leq n, \\ P(I_{n+1}) &= 1 - q. \end{aligned}$$

Therefore, the expected number of comparisons made by the algorithm for this probability distribution is given by the expected value of V:

$$\begin{aligned} \text{Avg}_A(n) = E(V) &= V(I_1)P(I_1) + \cdots + V(I_{n+1})P(I_{n+1}) \\ &= \frac{q}{n}(n + (n-1) + \cdots + 1) + (1-q)n \\ &= q\left(\frac{n+1}{2}\right) + (1-q)n. \end{aligned}$$

Let's observe a few things about the expected number of comparisons. If we know that X is in L, then $q = 1$. So the expectation is $(n + 1)/2$ comparisons. If we know that X is not in L, then $q = 0$, and the expectation is n comparisons. If X is in L and it occurs at the first position, then the algorithm takes n comparisons. So the worst case occurs for the two input cases I_{n+1} and I_1, and we have $W_A(n) = n$.

Approximations (Monte Carlo Method)

Sometimes it is not so easy to find a formula to solve a problem. In some of these cases we can find reasonable approximations by repeating some experiment many times and then observing the results. For example, suppose we have an irregular shape drawn on a piece of paper and we would like to know the area of the shape. The *Monte Carlo method* would have us randomly choose a large number of points on the paper. Then the area of the shape would be pretty close to the percentage of points that lie within the shape multiplied by the area of the paper.

The Monte Carlo method is useful in probability not only to check a calculated answer for a problem, but to find reasonable answers to problems for which we have no other answer. For example, a computer simulating thousands of repetitions of an experiment can give a pretty good approximation to the average outcome of the experiment.

Exercises

Permutations and Combinations

1. Evaluate each of the following expressions.

   a. $P(6, 6)$. b. $P(6, 0)$. c. $P(6, 2)$.
   d. $P(10, 4)$. e. $C(5, 2)$. f. $C(10, 4)$.

2. Let $S = \{a, b, c\}$. Write down the objects satisfying each of the following descriptions.

   a. All permutations of the three letters in S.
   b. All permutations consisting of two letters from S.
   c. All combinations of the three letters in S.
   d. All combinations consisting of two letters from S.
   e. All bag combinations consisting of two letters from S.

3. For each part of Exercise 2, write down the formula, in terms of P or C, for the number of objects requested.

4. Given the bag $B = [a, a, b, b]$, write down all the bag permutations of B, and verify with a formula that your answer is correct.

5. Find the number of ways to arrange the letters in each of the following words. Assume all letters are lowercase.

   a. Computer. b. Radar. c. States.
   d. Mississippi. e. Tennessee.

6. A *derangement* of a string is a permutation of the letters such that each letter changes its position. For example, a derangement of the string ABC

is BCA. But ACB is not a derangement of ABC, since A does not change position. Write down all derangements for each of the following strings.

a. A. b. AB. c. ABC. d. $ABCD$.

7. Suppose we want to build a code to represent 29 objects in which each object is represented as a binary string of length n, which consists of k 0's and m 1's, and $n = k + m$. Find n, k, and m, where n has the smallest possible value.

8. We wish to form a committee of seven people chosen from five Democrats, four Republicans, and six Independents. The committee will contain two Democrats, two Republicans, and three Independents. In how many ways can we choose the committee?

9. Each row of Pascal's triangle (Figure 5.4) has a largest number. Find a formula to describe which column contains the largest number in row n.

Discrete Probability

10. Suppose three fair coins are flipped. Find the probability for each of the following events.

a. Exactly one coin is a head. b. Exactly two coins are tails.
c. At least one coin is a head. d. At most two coins are tails.

11. Suppose a pair of dice are flipped. Find the probability for each of the following events.

a. The sum of the dots is 7.
b. The sum of the dots is even.
c. The sum of the dots is either 7 or 11.
d. The sum of the dots is at least 5.

12. A team has probability 2/3 of winning whenever it plays. Find each of the following probabilities that the team will win.

a. Exactly 4 out of 5 games.
b. At most 4 out of 5 games.
c. Exactly 4 out of 5 games given that it has already won the first 2 games of a 5-game series.

13. A baseball player's batting average is .250. Find each of the following probabilities that he will get hits.

a. Exactly 2 hits in 4 times at bat.
b. At least one hit in 4 times at bat.

14. A computer program uses one of three procedures for each piece of input. The procedures are used with probabilities 1/3, 1/2, and 1/6. Negative results are detected at rates of 10%, 20%, and 30% by the three procedures,

respectively. Suppose a negative result is detected. Find the probabilities that each of the procedures was used.

15. A commuter crosses one of three bridges, A, B, or C, to go home from work, crossing A with probability $1/3$, B with probability $1/6$, and C with probability $1/2$. The commuter arrives home by 6 p.m. 75%, 60%, and 80% of the time by crossing bridges A, B, and C, respectively. If the commuter arrives home by 6 p.m., find the probability that bridge A was used. Also find the probabilities for bridges B and C.

16. A student is chosen at random from a class of 80 students that has 20 honor students, 30 athletes, and 40 that are neither honor students nor athletes.
    a. What is the probability that the student selected is an athlete given that he or she is an honors student?
    b. What is the probability that the student selected is an honors student given that he or she is an athlete?
    c. Are the events "honors student" and "athlete" independent?

17. Suppose we have an algorithm that must perform 2,000 operations as follows: The first 1,000 operations are performed by a processor with a capacity of 100,000 operations per second. Then the second 1,000 operations are performed by a processor with a capacity of 200,000 operations per second. Find the average number of operations per second performed by the two processors to execute the 2,000 operations.

18. Consider each of the following lottery problems.
    a. Find the chances of winning a lottery that allows you to pick six numbers from the set $\{1, 2, \ldots, 49\}$.
    b. Suppose that a lottery consists of choosing a set of five numbers from the set $\{1, 2, \ldots, 49\}$. Suppose further that smaller prizes are given to people with four of the five winning numbers. What is the probability of winning a smaller prize?
    c. Suppose that a lottery consists of choosing a set of six numbers from the set $\{1, 2, \ldots, 49\}$. Suppose further that smaller prizes are given to people with four or five of the six winning numbers. What is the probability of winning a smaller prize?
    d. Find a formula for the probability of winning a smaller prize that goes with choosing k of the winning m numbers from the set $\{1, \ldots, n\}$, where $k < m < n$.

19. For each of the following problems, compute the expected value.
    a. The expected number of dots that show when a die is tossed.
    b. The expected score obtained by guessing all 100 questions of a true-false exam in which a correct answer is worth 1 point and an incorrect answer is worth $-1/2$ point.

Challenges

20. Test the birthday problem on a group of people.

21. Show that if S is a sample space and A is an event, then S and A are independent events. What about the independence of two events A and B that are disjoint?

22. Prove that if A and B are independent events, then so are the three pairs of events A and B', A' and B, and A' and B'.

23. Suppose an operating system must schedule the execution of n processes, where each process consists of k separate actions that must be done in order. Assume that any action of one process may run before or after any action of another process. How many execution schedules are possible?

24. Count the number of strings consisting of n 0's and n 1's such that each string is subject to the following restriction: As we scan a string from left to right, the number of 0's is never greater than the number of 1's. For example, the string 110010 is OK, but the string 100110 is not. *Hint:* Count the total number of strings of length $2n$ with n 0's and n 1's. Then try to count the number that are not OK, and subtract this number from the total number.

25. Given a nonempty finite set S with n elements, prove that there are $n!$ bijections from S to S.

26. (*Average-Case Analysis of Binary Search*).

    a. Assume that we have a sorted list of 15 elements, $x_1, x_2, \ldots, x_{15}$. Calculate the average number of comparisons made by a binary search algorithm to look for a key that may or may not be in the list. Assume that the key has probability 1/2 of being in the list and that each of the events "key $= x_i$" is equally likely for $1 \leq i \leq 15$.

    b. Generalize the problem to find a formula for the average number of comparisons used to look for a key in a sorted list of size $n = 2^k - 1$, where k is a natural number. Assume that the key has probability p of being in the list and that each of the events "key $= x_i$" is equally likely for $1 \leq i \leq n$. Test your formula with $n = 15$ and $p = 1/2$ to see that you get the same answer as part (a).

5.4 Solving Recurrences

Many counting problems result in answers that are expressed in terms of recursively defined functions. For example, any program that contains recursively defined procedures or functions will give rise to such expressions. Many of these expressions have closed forms that can simplify the counting process. So let's discuss how to find closed forms for such expressions.

Definition of Recurrence Relation

Any recursively defined function f with domain $\mathbb{N}$ that computes numbers is called a *recurrence* or a *recurrence relation.* When working with recurrences we often write f_n in place of $f(n)$. For example, the following definition is a recurrence:

$$\begin{aligned} r(0) &= 1 \\ r(n) &= 2r(n-1) + n. \end{aligned}$$

We can also write this recurrence in the following useful form:

$$\begin{aligned} r_0 &= 1 \\ r_n &= 2r_{n-1} + n. \end{aligned}$$

To *solve a recurrence* r we must find an expression for the general term r_n that is not recursive.

5.4.1 Solving Simple Recurrences

Let's start with some simple recurrences that can be solved without much fanfare. The recurrences we'll be considering have the following general form, where a_n and b_n denote either constants or expressions involving n but not involving r.

$$\begin{aligned} r_0 &= b_0, \\ r_n &= a_n r_{n-1} + b_n. \end{aligned} \tag{5.13}$$

We'll look at two similar techniques for solving these recurrences.

Solving by Substitution

One way to solve recurrences of the form (5.13) is by *substitution*, where we start with the definition for r_n and keep substituting for occurrences of r on the right side of the equation until we discover a pattern that allows us to skip ahead and eventually replace the basis r_0.

We'll demonstrate the substitution technique in general terms by solving (5.13). Note the patterns that emerge with each substitution for r.

$$\begin{aligned}
r_n &= a_n r_{n-1} + b_n \\
&= a_n \left(a_{n-1} r_{n-2} + b_{n-1}\right) + b_n && \text{(replace } r_{n-1} = a_{n-1} r_{n-2} + b_{n-1}\text{)} \\
&= a_n a_{n-1} r_{n-2} + a_n b_{n-1} + b_n && \text{(regroup)} \\
&= a_n a_{n-1} \left(a_{n-2} r_{n-3} + b_{n-3}\right) + a_n b_{n-1} + b_n \\
& && \text{(replace } r_{n-2} = a_{n-2} r_{n-3} + b_{n-2}\text{)} \\
&= a_n a_{n-1} a_{n-2} r_{n-3} + a_n a_{n-1} b_{n-3} + a_n b_{n-1} + b_n && \text{(regroup)} \\
&\ \ \vdots \\
&= a_n \cdots a_2 r_1 + a_n \cdots a_3 b_2 + \cdots + a_n b_{n-1} + b_n && \text{(regroup)} \\
&= a_n \cdots a_2 \left(a_1 r_0 + b_1\right) + a_n \cdots a_3 b_2 + \cdots + a_n b_{n-1} + b_n \\
& && \text{(replace } r_1 = a_1 r_0 + b_1\text{)} \\
&= a_n \cdots a_2 a_1 r_0 + a_n \cdots a_2 b_1 + a_n \cdots a_3 b_2 + \cdots + a_n b_{n-1} + b_n && \text{(regroup)} \\
&= a_n \cdots a_2 a_1 b_0 + a_n \cdots a_2 b_1 + a_n \cdots a_3 b_2 + \cdots + a_n b_{n-1} + b_n \\
& && \text{(replace } r_0 = b_0\text{)}
\end{aligned}$$

example

5.26 Solving by Substitution

We'll solve the following recurrence by substitution.

$$\begin{aligned}
r_0 &= 1, \\
r_n &= 2r_{n-1} + n.
\end{aligned}$$

Notice in the following solutions that we never multiply numbers. Instead we keep track of products to help us discover general patterns. Once we find a pattern we emphasize it with parentheses and exponents. Each line represents a substitution and regrouping of terms.

$$\begin{aligned}
r_n &= 2r_{n-1} + n \\
&= 2^2 r_{n-2} + 2\,(n-1) + n \\
&= 2^3 r_{n-3} + 2^2\,(n-2) + 2\,(n-1) + n \\
&\ \ \vdots \\
&= 2^{n-1} r_1 + 2^{n-2}\,(2) + 2^{n-2}\,(2) + \cdots + 2^2\,(n-2) + 2^1\,(n-1) + 2^0\,(n) \\
&= 2^n r_0 + 2^{n-1}\,(1) + 2^{n-2}\,(2) + \cdots + 2^2\,(n-2) + 2^1\,(n-1) + 2^0\,(n) \\
&= 2^n\,(1) + 2^{n-1}\,(1) + 2^{n-2}\,(2) + \cdots + 2^2\,(n-2) + 2^1\,(n-1) + 2^0\,(n).
\end{aligned}$$

Now we'll put it into closed form using (5.1), (5.2c), and (5.2d). Be sure you can see the reason for each step. We'll start by keeping the first term where it is and reversing the rest of the sum to get it in a nicer form.

$$\begin{aligned}
r_n &= 2^n(1) + n + 2(n-1) + 2^2(n-2) + \cdots + 2^{n-2}(2) + 2^{n-1}(1) \\
&= 2^n + \left[2^0(n) + 2^1(n-1) + 2^2(n-2) + \cdots + 2^{n-2}(2) + 2^{n-1}(1)\right] \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(group terms)} \\
&= 2^n + \sum_{i=0}^{n-1} 2^i(n-i) \\
&= 2^n + n\sum_{i=0}^{n-1} 2^i - \sum_{i=0}^{n-1} i2^i \\
&= 2^n + n(2^n - 1) - \left(2 - n2^n + (n-1)2^{n+1}\right) \\
&= 2^n(1 + n + n - 2n + 2) - n - 2 \\
&= 3(2^n) - n - 2.
\end{aligned}$$

Now check a few values of r_n to make sure that the sequence of numbers for the closed form and the recurrence are the same: $1, 3, 8, 19, 42, \ldots$.

end example

Solving by Cancellation

An alternative technique to solve recurences of the form (5.13) is by *cancellation*, where we start with the general equation for r_n. The term on the left side of each succeeding equation is the same as the term that contains r on the right side of the preceding equation. We normally write a few terms until a pattern emerges. The last equation always contains the basis element r_0 on the right side. Here is a sample.

$$\begin{aligned}
r_n &= a_n r_{n-1} + b_n \\
a_n r_{n-1} &= a_n a_{n-1} r_{n-2} + a_n b_{n-1} \\
a_n a_{n-1} r_{n-2} &= a_n a_{n-1} a_{n-2} r_{n-3} + a_n a_{n-1} b_{n-2} \\
&\ \ \vdots \\
a_n \cdots a_3 r_2 &= a_n \cdots a_2 r_1 + a_n \cdots a_3 b_2 \\
a_n \cdots a_2 r_1 &= a_n \cdots a_1 r_0 + a_n \cdots a_2 b_1
\end{aligned}$$

Now we add up the equations and observe that, except for r_n in the first equation, all terms on the left side of the remaning equations cancel with like terms on the right side of preceding equations. So the sum of the equations gives us the following formula for r_n, where we have replaced r_0 by its basis value b_0.

$$r_n = a_n \cdots a_1 b_0 + (b_n + a_n b_{n-1} + a_n a_{n-1} b_{n-2} + \cdots + a_n \cdots a_3 b_2 + a_n \cdots a_2 b_1)$$

So we get to the same place by either substitution or cancellation. Since mistakes are easy to make, it is nice to know that you can always check your solution against the original recurrence by testing. You can also give an induction proof that your solution is correct.

example

5.27 Solving by Cancellation

We'll solve the recurrence in Example 5.26 by cancellation:

$$\begin{aligned} r_0 &= 1, \\ r_n &= 2r_{n-1} + n. \end{aligned}$$

Starting with the general term, we obtain the following sequence of equations, where the term on the left side of a new equation is always the term that contains r from the right side of the preceding equation.

$$\begin{aligned} r_n &= 2r_{n-1} + n \\ 2r_{n-1} &= 2^2 r_{n-2} + 2\,(n-1) \\ 2^2 r_{n-2} &= 2^3 r_{n-3} + 2^2\,(n-2) \\ &\vdots \\ 2^{n-2} r_2 &= 2^{n-1} r_1 + 2^{n-2}\,(2) \\ 2^{n-1} r_1 &= 2^n r_0 + 2^{n-1}\,(1) \end{aligned}$$

Now add up all the equations, cancel the like terms, and replace r_0 by its value to get the following equation.

$$r_n = 2^n\,(1) + n + 2\,(n-1) + 2^2\,(n-2) + \cdots + 2^{n-2}\,(2) + 2^{n-1}\,(1).$$

Notice that, except for the ordering of terms, the solution is the same as the one obtained by substitution in Example 5.26.

end example

The Polynomial Problem

In Example 5.11 we found the number of arithmetic operations in a polynomial of degree n. By grouping terms of the polynomial we can reduce repeated multiplications. For example, here is the grouping when $n = 3$:

$$c_0 + c_1 x + c_2 x^2 + c_3 x^3 = c_0 + x\,(c_1 + x\,(c_2 + x\,(c_3))).$$

Notice that the expression on the left uses 9 operations while the expression on the right uses 6. The following function will evaluate a polynomial with terms grouped in this way, where C is the list of coefficients:

$$\text{poly}(C,\ x) = \text{if } C = \langle\ \rangle \text{ then } 0 \text{ else head}(C) + x * \text{poly}(\text{tail}(C),\ x).$$

For example, we'll evaluate the expression poly($\langle\ a,\ b,\ c,\ d\rangle$, x):

$$\begin{aligned}\text{poly}(\langle a,b,c,d\rangle,x) &= a + x * \text{poly}(\langle b,c,d\rangle,x)\\ &= a + x * (b + x * \text{poly}(\langle c,d\rangle,x))\\ &= a + x * (b + x * (c + x * \text{poly}(\langle d\rangle,x)))\\ &= a + x * (b + x * (c + x * (d + x * \text{poly}(\langle\ \rangle,x))))\\ &= a + x * (b + x * (c + d * 0))\end{aligned}$$

So there are 6 arithmetic operations performed by poly to evaluate a polynomial of degree 3. Let's figure out how many operations are performed to evalutate a polynomial of degree n. Let $a(n)$ denote the number of arithmetic operations performed by poly(C, x) when C has length n. If $n = 0$, then $C = \langle\ \rangle$ and

$$\text{poly}(C,\ x) = \text{poly}(\langle\ \rangle,\ x) = 0.$$

Therefore $a(0) = 0$. If $n > 0$, then $C \neq \langle\ \rangle$ and

$$\text{poly}(C,\ x) = \text{head}(C) + x * \text{poly}(\text{tail}(C),\ x).$$

This expression has two arithmetic operations plus the number of operations performed by poly(tail(C), x). Since tail(C) has $n - 1$ elements, it follows that poly(tail(C), x) performs $a(n - 1)$ operations. Therefore, for $n > 0$ we have a(n) = $a(n - 1) + 2$. So we have the following recursive definition:

$$\begin{aligned}a(0) &= 0\\ a(n) &= a(n-1) + 2.\end{aligned}$$

Writing it in subscripted form we have

$$\begin{aligned}a_0 &= 0\\ a_n &= a_{n-1} + 2\end{aligned}$$

It's easy to solve this recurrence by cancellation:

$$\begin{aligned}a_n &= a_{n-1} + 2\\ a_{n-1} &= a_{n-2} + 2\\ a_{n-2} &= a_{n-3} + 2\\ &\vdots\\ a_2 &= a_1 + 2\\ a_1 &= a_0 + 2\end{aligned}$$

Add up the equations and replace $a_0 = 0$ to obtain the solution

$$a_n = 2n.$$

This is quite a savings in the number of arithmetic operations to evaluate a polynomial of degree n. For example, if $n = 30$, then poly uses only 60 operations compared with 494 operations using the method discussed in Example 5.11.

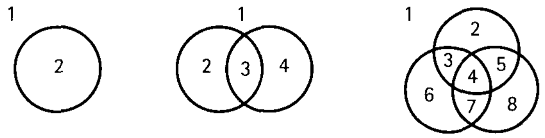


Figure 5.7 One, two, and three ovals.

The n-Ovals Problem

Suppose we are given the following sequence of three numbers:

$$2, 4, 8.$$

What is the next number in the sequence? The problem below might make you think about your answer.

The n-Ovals Problem
Suppose that n ovals (an oval is a closed curve that does not cross over itself) are drawn on the plane such that no three ovals meet in a point and each pair of ovals intersects in exactly two points. How many distinct regions of the plane are created by n ovals?

For example, the diagrams in Figure 5.7 show the cases for one, two, and three ovals.

If we let r_n denote the number of distinct regions of the plane for n ovals, then it's clear that the first three values are

$$\begin{aligned} r_1 &= 2, \\ r_2 &= 4, \\ r_3 &= 8. \end{aligned}$$

What is the value of r_4? Is it 16? Check it out. To find r_n, consider the following description: $n - 1$ ovals divide the region into r_{n-1} regions. The nth oval will meet each of the previous $n - 1$ ovals in $2(n - 1)$ points. So the nth oval will itself be divided into $2(n - 1)$ arcs. Each of these $2(n - 1)$ arcs splits some region in two. Therefore, we add $2(n - 1)$ regions to r_{n-1} to obtain r_n. This gives us the following recurrence.

$$\begin{aligned} r_1 &= 2, \\ r_n &= r_{n-1} + 2\,(n-1)\,. \end{aligned}$$

We'll solve it by the substitution technique:

$$\begin{aligned} r_n &= r_{n-1} + 2(n-1) \\ &= r_{n-2} + 2(n-2) + 2(n-1) \\ &\vdots \\ &= r_1 + 2(1) + \cdots + 2(n-2) + 2(n-1) \\ &= 2 + 2(1) + \cdots + 2(n-2) + 2(n-1). \end{aligned}$$

Now we can find a closed from for r_n.

$$\begin{aligned} r_n &= 2 + 2(1) + \cdots + 2(n-2) + 2(n-1) \\ &= 2 + 2(1 + 2 + \cdots (n-2) + (n-1)) \\ &= 2 + 2\sum_{i=1}^{n-1} i \\ &= 2 + 2\frac{(n-1)(n)}{2} \\ &= n^2 - n + 2. \end{aligned}$$

For example, we can use this formula to calculate $r_4 = 14$. Therefore, the sequence of numbers 2, 4, 8 could very well be the first three numbers in the following sequence for the n-ovals problem.

2, 4, 8, 14, 22, 32, 44, 62, 74, 92,....

5.4.2 Generating Functions

For some recurrence problems we need to find new techniques. For example, suppose we wish to find a closed form for the nth Fibonacci number F_n, which is defined by the recurrence system

$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_n &= F_{n-1} + F_{n-2} \quad (n \geq 2). \end{aligned}$$

We can't use substitution or cancellation with this system because F occurs twice on the right side of the general equation. This problem belongs to a large class of problems that need a more powerful technique.

The technique that we present comes from the simple idea of equating the coefficients of two polynomials. For example, suppose we have the following equation.

$$a + bx + cx^2 = 4 + 7x^2.$$

We can solve for a, b, and c by equating coefficients to yield $a = 4$, $b = 0$, and $c = 7$. We'll extend this idea to expressions that have infinitely many terms of the form $a_n x^n$ for each natural number n. Let's get to the definition.

Definition of Generating Function

The *generating function* for the infinite sequence $a_0, a_1, \ldots, a_n, \ldots$ is the following infinite expression, which is also called a formal power series or an infinite polynomial:

$$\begin{aligned} A(x) &= a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n + \cdots \\ &= \sum_{n=0}^{\infty} a_n x^n. \end{aligned}$$

Two generating functions may be added by adding the corresponding coefficients. Similarly, two generating functions may be multiplied by extending the rule for multiplying regular polynomials. In other words, multiply each term of one generating function by every term of the other generating function, and then add up all the results. Two generating functions are equal if their corresponding coefficients are equal.

We'll be interested in those generating functions that have closed forms. For example, let's consider the following generating function for the infinite sequence $1, 1, \ldots, 1, \ldots$:

$$\sum_{n=0}^{\infty} x^n.$$

This generating function is often called a *geometric series*, and its closed form is given by the following formula.

Geometric Series Generating Function **(5.14)**

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n.$$

To justify equation (5.14), multiply both sides of the equation by $1 - x$.

Using a Generating Function Formula

But how can we use this formula to solve recurrences? The idea, as we shall see, is to create an equation in which $A(x)$ is the unknown, solve for $A(x)$, and hope that our solution has a nice closed form. For example, if we find that

$$A(x) = \frac{1}{1-2x},$$

then we can rewrite it using (5.14) in the following way.

$$A(x) = \frac{1}{1-2x} = \frac{1}{1-(2x)} = \sum_{n=0}^{\infty} (2x)^n = \sum_{n=0}^{\infty} 2^n x^n.$$

Now we can equate coefficients to obtain the solution $a_n = 2^n$. In other words, the solution sequence is 1, 2, 4,..., 2^n, ...

Finding a Generating Function Formula

How do we obtain the closed form for $A(x)$? It's a four-step process, and we'll present it with an example. Suppose we want to solve the following recurrence:

$$\begin{aligned} a_0 &= 0, \\ a_1 &= 1, \\ a_n &= 5a_{n-1} - 6a_{n-2} \quad (n \geq 2). \end{aligned} \tag{5.15}$$

Step 1

Use the general equation in the recurrence to write an infinite polynomial with coefficients a_n. We start the index of summation at 2 because the general equation in (5.15) holds for $n \geq 2$. Thus we obtain the following equation:

$$\begin{aligned} \sum_{n=2}^{\infty} a_n x^n &= \sum_{n=2}^{\infty} (5a_{n-1} - 6a_{n-2})\, x^n \\ &= \sum_{n=2}^{\infty} 5a_{n-1} x^n - \sum_{n=2}^{\infty} 6a_{n-2} x^n \\ &= 5\sum_{n=2}^{\infty} a_{n-1} x^n - 6\sum_{n=2}^{\infty} a_{n-2} x^n \end{aligned} \tag{5.16}$$

We want to solve for $A(x)$ from this equation. Therefore, we need to transform each infinite polynomial in (5.16) into an expression containing $A(x)$. To do this, notice that the left-hand side of (5.16) can be written as

$$\begin{aligned} \sum_{n=2}^{\infty} a_n x^n &= A(x) - a_0 - a_1 x \\ &= A(x) - x \quad \text{(substitute for } a_0 \text{ and } a_1\text{)}. \end{aligned}$$

The first infinite polynomial on the right side of (5.16) can be written as

$$\begin{aligned}\sum_{n=2}^{\infty} a_{n+1}x^n &= \sum_{n=1}^{\infty} a_n x^{n+1} \quad \text{(by a change of indices)}\\ &= x\sum_{n=1}^{\infty} a_n x^n\\ &= x\left(A\left(x\right)-a_0\right)\\ &= xA\left(x\right).\end{aligned}$$

The second infinite polynomial on the right side of (5.16) can be written as

$$\begin{aligned}\sum_{n=2}^{\infty} a_{n-2}x^n &= \sum_{n=0}^{\infty} a_n x^{n+2} \quad \text{(by a change of indices)}\\ &= x^2\sum_{n=0}^{\infty} a_n x^n\\ &= x^2A\left(x\right).\end{aligned}$$

Thus Equation (5.16) can be rewritten in terms of $A(x)$ as

$$A(x) - x = 5xA(x) - 6x^2A(x). \tag{5.17}$$

Step 1 can often be done equationally by starting with the definition of $A(x)$ and continuing until an equation involving $A(x)$ is obtained. For this example the process goes as follows:

$$\begin{aligned}A\left(x\right) &= \sum_{n=0}^{\infty} a_n x^n\\ &= a_0 + a_1x + \sum_{n=2}^{\infty} a_n x^n\\ &= x + \sum_{n=2}^{\infty} a_n x^n\\ &= x + \sum_{n=2}^{\infty}\left(5a_{n-1} - 6a_{n-2}\right)x^n\\ &= x + 5\sum_{n=0}^{\infty} a_{n-1}x^n - 6\sum_{n=2}^{\infty} a_{n-2}x^n\\ &= x + 5x\left(A\left(x\right)-a_0\right) - 6x^2A\left(x\right)\\ &= x + 5xA\left(x\right) - 6x^2A\left(x\right).\end{aligned}$$

Step 2

Solve the equation for $A(x)$ and try to transform the result into an expression containing closed forms of known generating functions. We solve Equation (5.17) by isolating $A(x)$ as follows:

$$A(x)(1 - 5x + 6x^2) = x.$$

Therefore, we can solve for $A(x)$ and try to obtain known closed forms, which can then be replaced by generating functions:

$$\begin{aligned}
A(x) &= \frac{x}{1-5x+6x^2} \\
&= \frac{x}{(2x-1)(3x-1)} \\
&= \frac{1}{2x-1} - \frac{1}{3x-1} && \text{(partial fractions)} \\
&= -\frac{1}{1-2x} - \frac{1}{1-3x} && \text{(put into the form } \tfrac{1}{1-t}\text{)} \\
&= -\sum_{n=0}^{\infty}(2x)^n + \sum_{n=0}^{\infty}(3x)^n \\
&= -\sum_{n=0}^{\infty}2^n x^n + \sum_{n=0}^{\infty}3^n x^n \\
&= \sum_{n=0}^{\infty}(-2^n + 3^n)\,x^n.
\end{aligned}$$

Step 3

Equate coefficients, and obtain the result. In other words, we equate the original definition for $A(x)$ and the form of $A(x)$ obtained in Step 2:

$$\sum_{n=0}^{\infty} a_n x^n = \sum_{n=0}^{\infty}(-2^n + 3^n)\,x^n.$$

These two infinite polynomials are equal if and only if the corresponding coefficients are equal. Equating the coefficients, we obtain the following closed form for a_n:

$$a_n = 3^n - 2^n \quad \text{for } n \geq 0. \tag{5.18}$$

Step 4 (Check the answer)

To make sure that no mistakes were made in Steps 1 to 3, we should check to see whether (5.18) is the correct answer to (5.15). Since the recurrence has two basis cases, we'll start by verifying the special cases for $n = 0$ and $n = 1$. These cases are verified below:

$$\begin{aligned} a_0 &= 3^0 - 2^0 = 0, \\ a_1 &= 3^1 - 2^1 = 1. \end{aligned}$$

Now verify that (5.18) satisfies the general case of (5.15) for $n \geq 2$. We'll start on the right side of (5.15) and substitute (5.18) to obtain the left side of (5.15).

$$\begin{aligned} 5a_{n-1} - 6a_{n-2} &= 5\left(3^{n-1} - 2^{n-1}\right) - 6\left(3^{n-2} - 2^{n-2}\right) && \text{(substitution)} \\ &= 3^n - 2^n && \text{(simplification)} \\ &= a_n. \end{aligned}$$

An Aside on Partial Fractions

Let's recall a few facts about partial fractions. Suppose we are given the following quotient of two polynomials $p(x)$ and $q(x)$:

$$\frac{p(x)}{q(x)},$$

where the degree of $p(x)$ is less than the degree of $q(x)$. The first thing to do is factor $q(x)$ into a product of linear and/or quadratic polynomials that can't be factored further (say, over the real numbers). Therefore, each factor of $q(x)$ has one of the following forms:

$$ax + b \quad \text{or} \quad cx^2 + dx + e.$$

The *partial fraction* representation of

$$\frac{p(x)}{q(x)}$$

is a sum of terms, where each term in the sum is a quotient as follows:

Partial Fractions

1. If the linear polynomial $ax + b$ is repeated k times as a factor of $q(x)$, then add the following terms to the partial fraction representation, where $A_1, \ldots, A_k$ are constants to be determined:

$$\frac{A_1}{ax+b} + \frac{A_2}{(ax+b)^2} + \cdots + \frac{A_k}{(ax+b)^k}.$$

Continued ➧

➡ ➡

2. If the quadratic polynomial $cx^2 + dx + e$ is repeated k times as a factor of $q(x)$, then add the following terms to the partial fraction representation, where A_i and B_i are constants to be determined.

$$\frac{A_1x + B_1}{cx^2 + dx + e} + \frac{A_2x + B_2}{(cx^2 + dx + e)^2} + \cdots + \frac{A_kx + B_k}{(cx^2 + dx + e)^k}.$$

example

5.28 Sample Partial Fractions

Here are a few samples of partial fractions that can be obtained from the two rules.

$$\begin{aligned}
\frac{x-1}{x(x-2)(x+1)} &= \frac{A}{x} + \frac{B}{x-2} + \frac{C}{x+1}, \\
\frac{x^3-1}{x^2(x-2)^3} &= \frac{A}{x} + \frac{B}{x^2} + \frac{C}{x-2} + \frac{D}{(x-2)^2} + \frac{E}{(x-2)^3}, \\
\frac{x^2}{(x-1)(x^2+2x+1)} &= \frac{A}{x-1} + \frac{Bx+C}{x^2+2x+1}, \\
\frac{x}{(x-1)(x^2+1)^2} &= \frac{A}{x-1} + \frac{Bx+C}{x^2+1} + \frac{Dx+C}{(x^2+1)^2}.
\end{aligned}$$

end example

To determine the constants in a partial fraction representation, we can solve simultaneous equations. Suppose there are n constants to be found. Then we need to create n equations. To create an equation, pick some value for x, with the restriction that the value for x does not make any denominator zero. Do this for n distinct values for x. Then solve the resulting n equations. For example, in Step 2 of the generating function example we wrote down the following equalities.

$$\begin{aligned}
A(x) &= \frac{x}{1-5x+6x^2} \\
&= \frac{x}{(2x-1)(3x-1)} \\
&= \frac{1}{2x-1} - \frac{1}{3x-1}.
\end{aligned}$$

The last equality is the result of partial fractions. Here's how we got it. First we write the partial fraction representation

$$\frac{x}{(2x-1)(3x-1)} = \frac{A}{2x-1} + \frac{B}{3x-1}.$$

Then we create two equations in A and B by letting $x = 0$ and $x = 1$.

$$0 = -A - B$$
$$1/2 = A + (1/2)\,B$$

Solving for A and B, we get $A = 1$ and $B = -1$. This yields the desired equality

$$\frac{x}{(2x-1)(3x-1)} = \frac{1}{2x-1} - \frac{1}{3x-1}.$$

A Final Note on Partial Fractions

If the degree of the numerator $p(x)$ is greater than or equal to the degree of $q(x)$, then a simple division of $p(x)$ by $q(x)$ will yield an equation of the form

$$\frac{p(x)}{q(x)} = s(x) + \frac{p'(x)}{q'(x)}.$$

where the degree of $p'(x)$ is less than the degree of $q'(x)$. Then we can apply partial fractions to the quotient

$$\frac{p'(x)}{q'(x)}.$$

More Generating Functions

There are many useful generating functions. Since our treatment is not intended to be exhaustive, we'll settle for listing two more generating functions that have many applications.

Two More Useful Generating Functions

$$\frac{1}{(1-x)^{k+1}} = \sum_{n=0}^{\infty} \binom{k+n}{n} x^n \quad \text{for } k \in \mathbb{N}. \tag{5.19}$$

$$(1+x)^r = \sum_{n=0}^{\infty} \left(\frac{r(r-1)\cdots(r-n+1)}{n!}\right) x^n \quad \text{for } r \in \mathbb{R}. \tag{5.20}$$

The numerator of the coefficient expression for the nth term in (5.20) contains a product of n numbers. When $n = 0$, we use the convention that a vacuous product—of zero numbers—has the value 1. Therefore the 0th term of (5.20) is $1/0! = 1$. So the first few terms of (5.20) look like the following:

$$(1+x)^r = 1 + rx + \frac{r(r-1)}{2}x^2 + \frac{r(r-1)(r-2)}{6}x^3 + \cdots.$$

The Problem of Parentheses

Suppose we want to find the number of ways to parenthesize the expression

$$t_1 + t_2 + \cdots + t_{n-1} + t_n \tag{5.21}$$

so that a parenthesized form of the expression reflects the process of adding two terms. For example, the expression $t_1 + t_2 + t_3 + t_4$ has several different forms as shown in the following expressions:

$$\begin{gathered}((t_1 + t_2) + (t_3 + t_4)) \\ (t_1 + (t_2 + (t_3 + t_4))) \\ (t_1 + ((t_2 + t_3) + t_4)) \\ \vdots\end{gathered}$$

To solve the problem, we'll let b_n denote the total number of possible parenthesizations for an n-term expression. Notice that if $1 \le k \le n - 1$, then we can split the expression (5.21) into two subexpressions as follows:

$$t_1 + \cdots + t_{n-k} \quad \text{and} \quad t_{n-k+1} + \cdots + t_n. \tag{5.22}$$

So there are $b_{n-k}b_k$ ways to parenthesize the expression (5.21) if the final + is placed between the two subexpressions (5.22). If we let k range from 1 to $k - 1$, we obtain the following formula for b_n when $n \ge 2$:

$$b_n = b_{n-1}b_1 + b_{n-2}b_2 + \cdots + b_2b_{n-2} + b_1b_{n-1}. \tag{5.23}$$

But we need $b_1 = 1$ for (5.23) to make sense. It's OK to make this assumption because we're concerned only about expressions that contain at least two terms. Similarly, we can let $b_0 = 0$. So we can write down the recurrence to describe the solution as follows:

$$\begin{aligned} b_0 &= 0, \\ b_1 &= 1, \\ b_n &= b_nb_0 + b_{n-1}b_1 + \cdots + b_1b_{n-1} + b_0b_n \quad (n \ge 2). \end{aligned} \tag{5.24}$$

Notice that this system cannot be solved by substitution or cancellation. Let's try generating functions. Let $B(x)$ be the generating function for the sequence

$$b_0,\ b_1, \ldots,\ b_n,\ \ldots.$$

So $B(x) = \sum_{n=0}^{\infty} b_nx^n$. Now let's try to apply the four-step procedure for generating functions. First we use the general equation in the recurrence to introduce the partial (since $n \ge 2$) generating function

$$\sum_{n=2}^{\infty} b_nx^n = \sum_{n=2}^{\infty} (b_nb_0 + b_{n-1}b_1 + \cdots + b_1b_{n-1} + b_0b_n)\, x^n. \tag{5.25}$$

Now the left-hand side of (5.25) can be written in terms of $B(x)$:

$$\begin{aligned}\sum_{n=2}^{\infty} b_n x^n &= B(x) - b_1 x - b_0 \\ &= B(x) - x \quad (\text{since } b_0 = 0 \text{ and } b_1 = 1).\end{aligned}$$

Before we discuss the right hand-side of Equation (5.25), notice that we can write the product

$$\begin{aligned}B(x)\,B(x) &= \left(\sum_{n=0}^{\infty} b_n x^n\right)\left(\sum_{n=0}^{\infty} b_n x^n\right) \\ &= \sum_{n=0}^{\infty} c_n x^n,\end{aligned}$$

where $c_0 = b_0\ b_0$ and, for $n > 0$,

$$c_n = b_n\ b_0 + b_{n-1}\ b_1 + \cdots + b_1\ b_{n-1} + b_0\ b_n.$$

So the right-hand side of Equation (5.25) can be written as

$$\begin{aligned}&\sum_{n=2}^{\infty} \left(b_n b_0 + b_{n-1} b_1 + \cdots + b_1 b_{n-1} + b_0 b_n\right) x^n \\ &\qquad = B(x)\,B(x) - b_0 b_0 - \left(b_1 b_0 + b_0 b_1\right) x \\ &\qquad = B(x)\,B(x) \quad (\text{since } b_0 = 0).\end{aligned}$$

Now Equation (5.25) can be written in simplified form as

$$B(x) - x = B(x)\,B(x) \quad \text{or} \quad B(x)^2 - B(x) + x = 0.$$

Now, thinking of $B(x)$ as the unknown, the equation is a quadratic equation with two solutions:

$$B(x) = \frac{1 \pm \sqrt{1 - 4x}}{2}.$$

Notice that $\sqrt{1-4x}$ is the closed form for generating a function obtained from (5.20), where $r = \frac{1}{2}$. Thus we can write

$$\begin{aligned}\sqrt{1-4x} &= (1+(-4x))^{\frac{1}{2}} \\ &= \sum_{n=0}^{\infty} \frac{\frac{1}{2}\left(\frac{1}{2}-1\right)\left(\frac{1}{2}-2\right)\cdots\left(\frac{1}{2}-n+1\right)}{n!}(-4x)^n \\ &= \sum_{n=0}^{\infty} \frac{\frac{1}{2}\left(-\frac{1}{2}\right)\left(-\frac{3}{2}\right)\cdots\left(-\frac{2n-3}{2}\right)}{n!}(-2)^n\, 2^n x^n \\ &= 1+\sum_{n=1}^{\infty} \frac{(-1)(1)(3)\cdots(2n-3)}{n!} 2^n x^n \\ &= 1+\sum_{n=1}^{\infty}\left(-\frac{2}{n}\right)\binom{2n-2}{n-1}x^n.\end{aligned}$$

Expansion of the last equality is left as an exercise. Notice that, for $n \geq 1$, the coefficient of x^n is negative in this generating function. In other words, the nth term ($n \geq 1$) of the generating function, for $\sqrt{1-4x}$ always has a negative coefficient. Since we need positive values for b_n, we must choose the following solution of our quadratic equation:

$$B(x) = \frac{1}{2} - \frac{1}{2}\sqrt{1-4x}.$$

Putting things together, we can write our desired generating function as follows:

$$\begin{aligned}\sum_{n=0}^{\infty} b_n x^n = B(x) &= \frac{1}{2} - \frac{1}{2}\sqrt{1-4x} \\ &= \frac{1}{2} - \frac{1}{2}\left\{1+\sum_{n=1}^{\infty}\left(-\frac{2}{n}\right)\binom{2n-2}{n-1}x^n\right\} \\ &= 0+\sum_{n=1}^{\infty}\frac{1}{n}\binom{2n-2}{n-1}x^n.\end{aligned}$$

Now we can finish the job by equating coefficients to obtain the following solution:

$$b_n = \text{if } n=0 \text{ then } 0 \text{ else } \frac{1}{n}\binom{2n-2}{n-1}.$$

The Problem of Binary Trees

Suppose we want to find, for any natural number n, the number of structurally distinct binary trees with n nodes. Let b_n denote this number. We can figure out a few values by experiment. For example, since there is one empty binary tree and one binary tree with a single node, we have $b_0 = 1$ and $b_1 = 1$. It's also easy to see that $b_2 = 2$, and for $n = 3$ we see after a few minutes that $b_3 = 5$.

Let's consider b_n for $n \geq 1$. A tree with n nodes has a root and two subtrees whose combined nodes total $n - 1$. For each k in the interval $0 \leq k \leq n - 1$ there are b_k left subtrees of size k and b_{n-1-k} distinct right subtrees of size $n - 1 - k$. So for each k there are $b_k b_{n-1-k}$ distinct binary trees with n nodes. Therefore, the number b_n of binary trees can be given by the sum of these products as follows:

$$b_n = b_0 b_{n-1} + b_1 b_{n-2} + \cdots + b_k b_{n-k} + \cdots + b_{n-2}\, b_1 + b_{n-1} b_0.$$

Now we can write down the recurrence to describe the solution as follows:

$$\begin{aligned} &b_0 = 1, \\ &b_n = b_0 b_{n-1} + b_1 b_{n-2} + \cdots + b_k b_{n-k} + \cdots + b_{n-2} b_1 + b_{n-1} b_0 \quad (n \geq 1) \end{aligned}$$

Notice that this system cannot be solved by cancellation or substitution. Let's try generating functions. Let $B(x)$ be the generating function for the sequence

$$b_0,\ b_1, \ldots,\ b_n,\ \ldots.$$

So $B(x) = \sum_{n=0}^{\infty} b_n x^n$. Now let's try to apply the four-step procedure for generating functions. First we use the general equation in the recurrence to introduce the partial (since $n \geq 1$) generating function

$$\sum_{n=1}^{\infty} b_n x^n = \sum_{n=1}^{\infty} \left(b_0 b_{n-1} + b_1 b_{n-2} + \cdots + b_{n-2} b_1 + b_{n-1} b_0\right) x^n. \tag{5.26}$$

Now the left-hand side of (5.26) can be written in terms of $B(x)$.

$$\begin{aligned} \sum_{n=1}^{\infty} b_n x^n &= B(x) - b_0 \\ &= B(x) - 1 \quad (\text{since } b_0 = 1). \end{aligned}$$

Before we discuss the right hand-side of Equation (5.26), notice that we can write the product

$$\begin{aligned} B(x)\, B(x) &= \left(\sum_{n=0}^{\infty} b_n x^n\right) \left(\sum_{n=0}^{\infty} b_n x^n\right) \\ &= \sum_{n=0}^{\infty} c_n x^n, \end{aligned}$$

where $c_0 = b_0\ b_0$ and for $n > 0$,

$$c_n = b_0 b_n + b_1 b_{n-1} + \cdots + b_{n-1} b_1 + b_n b_0$$

So the right-hand side of Equation (5.26) can be written as

$$\begin{aligned}
&\sum_{n=1}^{\infty}\left(b_0b_{n-1}+b_1b_{n-2}+\cdots+b_{n-2}b_1+b_{n-1}b_0\right)x^n\\
&\qquad=\sum_{n=0}^{\infty}\left(b_0b_n+b_1b_{n-1}+\cdots+b_{n-1}b_1+b_nb_0\right)x^{n+1}\\
&\qquad=x\sum_{n=0}^{\infty}\left(b_0b_n+b_1b_{n-1}+\cdots+b_{n-1}b_1+b_nb_0\right)x^n\\
&\qquad=xB\left(x\right)B\left(x\right)
\end{aligned}$$

Now Equation (5.26) can be written in simplified form as

$$B\left(x\right)-1=xB\left(x\right)B\left(x\right)\quad\text{or}\quad xB\left(x\right)^2-B\left(x\right)+1=0.$$

Now, thinking of $B(x)$ as the unknown, the equation is a quadratic equation with two solutions:

$$B\left(x\right)=\frac{1\pm\sqrt{1-4x}}{2x}.$$

Notice that $\sqrt{1-4x}$ is the closed form for generating a function obtained from (5.20), where $r=\frac{1}{2}$. Thus we can write

$$\begin{aligned}
\sqrt{1-4x}&=\left(1+\left(-4x\right)\right)^{\frac{1}{2}}\\
&=\sum_{n=0}^{\infty}\frac{\frac{1}{2}\left(\frac{1}{2}-1\right)\left(\frac{1}{2}-2\right)\cdots\left(\frac{1}{2}-n+1\right)}{n!}\left(-4x\right)^n\\
&=\sum_{n=0}^{\infty}\frac{\frac{1}{2}\left(-\frac{1}{2}\right)\left(-\frac{3}{2}\right)\cdots\left(-\frac{2n-3}{2}\right)}{n!}\left(-2\right)^n2^nx^n\\
&=1+\sum_{n=1}^{\infty}\frac{\left(-1\right)\left(1\right)\left(3\right)\cdots\left(2n-3\right)}{n!}2^nx^n\\
&=1+\sum_{n=1}^{\infty}\left(-\frac{2}{n}\right)\binom{2n-2}{n-1}x^n.
\end{aligned}$$

Notice that for $n\geq 1$ the coefficient of x^n is negative in this generating function. In other words, the nth term ($n\geq 1$) of the generating function for $\sqrt{1-4x}$ always has a negative coefficient. Since we need positive values for b_n, we must choose the following solution of our quadratic equation:

$$B\left(x\right)=\frac{1-\sqrt{1-4x}}{2x}.$$

Putting things together, we can write our desired generating function as follows:

$$\begin{aligned}\sum_{n=0}^{\infty} b_n x^n = B(x) = \frac{1-\sqrt{1-4x}}{2x} &= \frac{1}{2x}\left(1-\sqrt{1-4x}\right)\\ &= \frac{1}{2x}\sum_{n=1}^{\infty}\left(\frac{2}{n}\right)\binom{2n-2}{n-1}x^n\\ &= \sum_{n=1}^{\infty}\frac{1}{n}\binom{2n-2}{n-1}x^{n+1}\\ &= \sum_{n=0}^{\infty}\frac{1}{n+1}\binom{2n}{n}x^n.\end{aligned}$$

Now we can finish the job by equating coefficients to obtain

$$b_n = \frac{1}{n+1}\binom{2n}{n}.$$

Exercises

Simple Recurrences

1. Solve each of the following recurrences by the substitution technique and the cancellation technique. Put each answer in closed form (no ellipsis allowed).

   a. $a_1 = 0,$ $a_n = a_{n-1} + 4.$

   b. $a_1 = 0,$ $a_n = a_{n-1} + 2n.$

   c. $a_0 = 1,$ $a_n = 2a_{n-1} + 3.$

2. For each of the following definitions, find a recurrence to describe the number of times the cons operation :: is called. Solve each recurrence.

```
a. cat(L, M) = if L = ⟨ ⟩ then M else head(L) :: cat(tail(L), M).
b. dist(x, L) = if L = ⟨ ⟩ then ⟨ ⟩
                else (x :: head(L) :: ⟨ ⟩) :: dist(x, tail(L)).
c. power(L) = if L = ⟨ ⟩ then return ⟨ ⟩ :: ⟨ ⟩
              else
                  A := power(tail(L));
                  B := dist(head(L), A);
                  C := map(::, B);
                  return cat(A, C)
              fi
```

3. (*Towers of Hanoi*). The *Towers of Hanoi* puzzle was invented by Lucas in 1883. It consists of three stationary pegs with one peg containing a stack of

n disks that form a tower (each disk has a hole in the center for the peg) in which each disk has a smaller diameter than the disk below it. The problem is to move the tower to one of the other pegs by transferring one disk at a time from one peg to another peg, no disk ever being placed on a smaller disk. Find the minimum number of moves H_n to do the job.

Hint: It takes 0 moves to transfer a tower of 0 disks and 1 move to transfer a tower of 1 disk. So $H_0 = 0$ and $H_1 = 1$. Try it out for $n = 2$ and $n = 3$ to get the idea. Then try to find a recurrence relation for the general term H_n as follows: Move the tower consisting of the top $n - 1$ disks to the nonchosen peg; then move the bottom disk to the chosen peg; then move the tower of $n - 1$ disks onto the chosen peg.

4. (*Diagonals in a Polygon*). A diagonal in a polygon is a line from one vertex to another nonadjacent vertex. For example, a triangle doesn't have any diagonals because each vertex is adjacent to the other vertices. Find the number of diagonals in an n-sided polygon, where $n \geq 3$.

5. (*The n-Lines Problem*). Find the number of regions in a plane that are created by n lines, where no two lines are parallel and where no more than two lines intersect at any point.

Generating Functions

6. Given the generating function $A(x) = \sum_{n=0}^{\infty} a_n x^n$, find a closed form for the general term a_n for each of the following representations of $A(x)$.

   a. $A(x) = \frac{1}{x-2} - \frac{2}{3x+1}$.
   b. $A(x) = \frac{1}{2x+1} + \frac{3}{x+6}$.
   c. $A(x) = \frac{1}{3x-2} - \frac{1}{(1-x)^2}$.

7. Use generating functions to solve each of the following recurrences.

   a. $a_0 = 0$,
   $a_1 = 4$,
   $a_n = 2a_{n-1} + 3a_{n-2} \quad (n \geq 2)$.
   b. $a_0 = 0$,
   $a_1 = 1$,
   $a_n = 7a_{n-1} - 12a_{n-2} \quad (n \geq 2)$.
   c. $a_0 = 0$,
   $a_1 = 1$,
   $a_2 = 1$,
   $a_n = 2a_{n-1} + a_{n-2} - 2a_{n-3} \quad (n \geq 3)$.

8. Use generating functions to solve each recurrence in Exercise 1. For those recurrences that do not have an a_0 term, assume that $a_0 = 0$.

Proofs and Challenges

9. Prove in two different ways that the following equation holds for all positive integers n, as indicated:

$$\frac{(1)(1)(3)\cdots(2n-3)}{n!}2^n = \frac{2}{n}\binom{2n-2}{n-1}.$$

   a. Use induction.
   b. Transform the left side into the right side by "inserting" the missing even numbers in the numerator.

10. Find a closed form for the nth Fibonacci number defined by the following recurrence system.

$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_n &= F_{n-1} + F_{n-2} \quad (n \geq 2). \end{aligned}$$

5.5 Comparing Rates of Growth

Sometimes it makes sense to approximate the number of steps required to execute an algorithm because of the difficulty involved in finding a closed form for an expression or the difficulty in evaluating an expression. To approximate one function with another function, we need some way to compare them. That's where "rate of growth" comes in. We want to give some meaning to statements like "f has the same growth rate as g" and "f has a lower growth rate than g."

For our purposes we will consider functions whose domains and codomains are subsets of the real numbers. We'll examine the asymptotic behavior of two functions f and g by comparing $f(n)$ and $g(n)$ for large positive values of n (i.e., as n approaches infinity).

5.5.1 Big Theta

Let's begin by discussing the meaning of the statement "f has the same growth rate as g."

A function f has the *same growth rate* as g (or f has the *same order* as g) if we can find a number m and two positive constants c and d such that

$$c\,|g(n)| \leq |f(n)| \leq d\,|g(n)| \quad \text{for all } n \geq m. \tag{5.27}$$

In this case we write $f(n) = \Theta(g(n))$ and say that $f(n)$ is *big theta* of $g(n)$.

It's easy to verify that the relation "has the same growth rate as" is an equivalence relation. In other words, the following three properties hold for all functions.

$$f(n) = \Theta(f(n)).$$
$$\text{If } f(n) = \Theta(g(n)), \text{ then } g(n) = \Theta(f(n)).$$
$$\text{If } f(n) = \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)), \text{ then } f(n) = \Theta(h(n)).$$

If $f(n) = \Theta(g(n))$ and we also know that $g(n) \neq 0$ for all $n \geq m$, then we can divide the inequality (5.27) by $g(n)$ to obtain

$$c \leq \left|\frac{f(n)}{g(n)}\right| \leq d \quad \text{for all } n \geq m.$$

This inequality gives us a better way to think about "having the same growth rate." It tells us that the ratio of the two functions is always within a fixed bound beyond some point. We can always take this point of view for functions that count the steps of algorithms because they are positive valued.

Now let's see whether we can find some functions that have the same growth rate. To start things off, suppose f and g are proportional. This means that there is a nonzero constant c such that $f(n) = cg(n)$ for all n. In this case, definition (5.27) is satisfied by letting $d = c$. Thus we have the following statement.

Proportionality **(5.28)**
If two functions f and g are proportional, then $f(n) = \Theta(g(n))$.

example

5.29 The Log Function

Recall that log functions with different bases are proportional. In other words, if we have two bases $a > 1$ and $b > 1$, then

$$\log_a n = (\log_a b)(\log_b n) \quad \text{for all } n > 0.$$

So we can disregard the base of the log function when considering rates of growth. In other words, we have

$$\log_a n = \Theta(\log_b n). \tag{5.29}$$

end example

It's interesting to note that two functions can have the same growth rate without being proportional. Here's an example.

example

5.30 Polynomials of the Same Degree

Let's show that $n^2 + n$ and n^2 have the same growth rate. The following inequality is true for all $n \geq 1$:

$$1n^2 \leq n^2 + n \leq 2n^2.$$

Therefore, $n^2 + n = \Theta(n^2)$.

end example

The following theorem gives us a nice tool for showing that two functions have the same growth rate.

Theorem **(5.30)**

$$\text{If } \lim_{n\to\infty} \frac{f(n)}{g(n)} = c \quad \text{where } c \neq 0 \text{ and } c \neq \infty, \text{ then } f(n) = \Theta(g(n)).$$

For example, the quotient $(25n^2 + n)/n^2$ approaches 25 as n approaches infinity. Therefore we can say that $25n^2 + n = \Theta(n^2)$.

We should note that the limit in (5.30) is not a necessary condition for $f(n) = \Theta(g(n))$. For example, suppose we let f and g be the two functions

$$f(n) = \text{if } n \text{ is odd then 2 else 4,}$$
$$g(n) = 2.$$

We can write $1 \cdot g(n) \leq f(n) \leq 2 \cdot g(n)$ for all $n \geq 1$. Therefore $f(n) = \Theta(g(n))$. But the quotient $f(n)/g(n)$ alternates between the two values 1 and 2. Therefore the limit of the quotient does not exist. Still the limit test (5.30) will work for the majority of functions that occur in analyzing algorithms.

Approximations can be quite useful for those of us who can't remember formulas that we don't use all the time. For example, the first four of the following approximations are the summation formulas from (5.2) written in terms of Θ.

Some Approximations

$$\sum_{i=1}^{n} i = \Theta(n^2). \tag{5.31}$$

$$\sum_{i=1}^{n} i^2 = \Theta(n^3). \tag{5.32}$$

Continued ➡

➡ ➡

$$\text{If } a \neq 1, \text{ then } \sum_{i=0}^{n} a^i = \Theta\left(a^{n+1}\right). \tag{5.33}$$

$$\text{If } a \neq 1, \text{ then } \sum_{i=0}^{n} ia^i = \Theta\left(na^{n+1}\right). \tag{5.34}$$

$$\sum_{i=1}^{n} i^k = \Theta\left(n^{k+1}\right). \tag{5.35}$$

Notice that (5.31) and (5.32) are special cases of (5.35).

example 5.31 A Worst-Case Lower Bound for Sorting

Let's clarify a statement that we made in Example 5.13. We showed that $\lceil \log_2 n! \rceil$ is the worst-case lower bound for comparison sorting algorithms. But log $n!$ is hard to calculate for even modest values of n. We stated that $\lceil \log_2 n! \rceil$ is approximately equal to $n \log_2 n$. Now we can make the following statement:

$$\log n! = \Theta\left(n \log n\right). \tag{5.36}$$

To prove this statement, we'll find some bounds on log $n!$ as follows:

$$\begin{aligned}
\log n! &= \log n + \log(n-1) + \cdots + \log 1 \\
&\le \log n + \log n + \cdots + \log n && (n \text{ terms}) \\
&= n \log n.
\end{aligned}$$

$$\begin{aligned}
\log n! &= \log n + \log(n-1) + \cdots + \log 1 \\
&\ge \log n + \log(n-1) + \cdots + \log(\lceil n/2 \rceil) && (\lceil n/2 \rceil \text{ terms}) \\
&\ge \log \lceil n/2 \rceil + \cdots + \log \lceil n/2 \rceil && (\lceil n/2 \rceil \text{ terms}) \\
&= \lceil n/2 \rceil \log \lceil n/2 \rceil \\
&\ge (n/2) \log(n/2).
\end{aligned}$$

So we have the inequality:

$$(n/2)\log(n/2) \le \log n! \le n \log n.$$

It's easy to see (i.e., as an exercise) that if $n > 4$, then $(1/2) \log n < \log (n/2)$. Therefore, we have the following inequality for $n > 4$:

$$(1/4)\ (n \log n) \le (n/2) \log (n/2) \le \log n! \le n \log n.$$

So there are nonzero constants 1/4 and 1 and the number 4 such that

$$(1/4)\ (n \log n) \le \log n! \le (1)(n \log n) \text{ for all } n > 4.$$

This tells us that $\log n! = \Theta(n \log n)$.

end example

An important approximation to $n!$ is *Stirling's formula*—named for the mathematician James Stirling (1692–1770)—which is written as

$$n! = \Theta\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right). \tag{5.37}$$

Let's see how we can use big theta to discuss the approximate performance of algorithms. For example, the worst-case performance of the binary search algorithm is $\Theta(\log n)$ because the actual value is $1 + \lfloor \log_2 n \rfloor$. Both the average and worst-case performances of a linear sequential search are $\Theta(n)$ because the average number of comparisons is $(n+1)/2$ and the worst-case number of comparisons is n.

For sorting algorithms that sort by comparison, the worst-case lower bound is $\lceil \log_2 n! \rceil = \Theta(n \log n)$. Many sorting algorithms, like the simple sort algorithm in Example 5.12, have worst-case performance of $\Theta(n^2)$. The "dumbSort" algorithm, which constructs a permutation of the given list and then checks to see whether it is sorted, may have to construct all possible permutations before it gets the right one. Thus dumbSort has worst-case performance of $\Theta(n!)$. An algorithm called "heapsort" will sort any list of n items using at most $2n \log_2 n$ comparisons. So heapsort is a $\Theta(n \log n)$ algorithm in the worst case.

5.5.2 Little Oh

Now let's discuss the meaning of the statement "f has a lower growth rate than g."

A function f has a *lower growth rate* than g (or f has *lower order* than g) if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0. \tag{5.38}$$

In this case we write $f(n) = o\,(g(n))$ and say that f is *little oh* of g.

For example, the quotient n/n^2 approaches 0 as n goes to infinity. Therefore, $n = o(n^2)$, and we can say that n has lower order than n^2. For another example, if a and b are positive numbers such that $a < b$, then $a^n = o(b^n)$. To see this, notice that the quotient approaches 0 as n approaches infinity because $0 < a/b < 1$.

For those readers familiar with derivatives, the evaluation of limits can often be accomplished by using L'Hôpital's rule.

Theorem **(5.39)**

If $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = \infty$ or $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = 0$ and f and g are differentiable beyond some point, then

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}.$$

example

5.32 Different Orders

We'll show that log $n = o(n)$. Since both n and log n approach infinity as n approaches infinity, we can apply (5.39) to (log $n)/n$. Since we can write log $n =$ (log e)(log$_e$ n), it follows that the derivative of log n is (log e)($1/n$). Therefore, we obtain the following equations:

$$\lim_{n\to\infty} \frac{\log n}{n} = \lim_{n\to\infty} \frac{(\log e)\,(1/n)}{1} = 0.$$

So log n has lower order than n, and we can write log $n = o(n)$.

end example

Let's list a hierarchy of some familiar functions according to their growth rates, where $f(n) \prec g(n)$ means that $f(n) = o(g(n))$:

$$1 \prec \log n \prec n \prec n \log n \prec n^2 \prec n^3 \prec 2^n \prec 3^n \prec n! \prec n^n. \qquad (5.40)$$

This hierarchy can help us compare different algorithms. For example, we would certainly choose an algorithm with running time $\Theta(\log n)$ over an algorithm with running time $\Theta(n)$.

5.5.3 Big Oh and Big Omega

Now let's look at a notation that gives meaning to the statement "the growth rate of f is bounded above by the growth rate of g." The standard notation to describe this situation is

$$f(n) = O(g(n)), \qquad (5.41)$$

which we read as $f(n)$ is *big oh* of $g(n)$. The precise meaning of the notation $f(n) = O(g(n))$ is given by the following definition.

The Meaning of Big Oh **(5.42)**

The notation $f(n) = O(g(n))$ means that there are positive numbers c and m such that

$$|f(n)| \leq c\,|g(n)| \text{ for all } n \geq m.$$

example **5.33 Comparing Polynomials**

We'll show that $n^2 = O(n^3)$ and $5n^3 + 2n^2 = O(n^3)$. Since $n^2 \leq 1n^3$ for all $n \geq 1$, it follows that $n^2 = O(n^3)$. Since $5n^3 + 2n^2 \leq 7n^3$ for all $n \geq 1$, it follows that $5n^3 + 2n^2 = O(n^3)$.

end example

Now let's go the other way. We want a notation that gives meaning to the statement "the growth rate of f is bounded below by the growth rate of g." The standard notation to describe this situation is

$$f(n) = \Omega(g(n)), \tag{5.43}$$

which we can read as $f(n)$ is *big omega* of $g(n)$. The precise meaning of the notation $f(n) = \Omega\ (g(n))$ is given by the following definition.

The Meaning of Big Omega **(5.44)**

The notation $f(n) = \Omega(g(n))$ means that there are positive numbers c and m such that

$$|f(n)| \geq c\,|g(n)| \text{ for all } n \geq m.$$

example **5.34 Comparing Polynomials**

We'll show that $n^3 = \Omega(n^2)$ and $3n^2 + 2n = \Omega(n^2)$. Since $n^3 \geq 1n^2$ for all $n \geq 1$, it follows that $n^3 = \Omega(n^2)$. Since $3n^2 + 2n \geq 1n^2$ for all $n \geq 1$, it follows that $3n^2 + 2n = \Omega(n^2)$.

end example

Let's see how we can use the terms that we've defined so far to discuss algorithms. For example, suppose we have constructed an algorithm A to solve some problem P. Suppose further that we've analyzed A and found that it takes $5n^2$ operations in the worst case for an input of size n. This allows us to make a few general statements. First, we can say that the worst-case performance of A

is $\Theta(n^2)$. Second, we can say that an optimal algorithm for P, if one exists, must have a worst-case performance of $O(n^2)$. In other words, an optimal algorithm for P must do no worse than our algorithm A.

Continuing with our example, suppose some good soul has computed a worst-case theoretical lower bound of $\Theta(n \log n)$ operations for any algorithm that solves P. Then we can say that an optimal algorithm, if one exists, must have a worst-case performance of $\Omega(n \log n)$. In other words, an optimal algorithm for P can do no better than the given lower bound of $\Theta(n \log n)$.

Before we leave our discussion of approximate optimality, let's look at some other ways to use the symbols. The four symbols Θ, o, O, and Ω can also be used to represent terms within an expression. For example, the equation

$$h(n) = 4n^3 + O(n^2)$$

means that $h(n)$ equals $4n^3$ plus a term of order at most n^2. When used as part of an expression, big oh is the most popular of the four symbols because it gives a nice way to concentrate on those terms that contribute the most muscle.

We should also note that the four symbols Θ, o, O, and Ω can be formally defined to represent sets of functions. In other words, for a function g we define the following four sets:

$\Theta(g)$ is the set of functions with the same order as g;

$o(g)$ is the set of functions with lower order than g;

$O(g)$ is the set of functions of order bounded above by that of g;

$\Omega(g)$ is the set of functions of order bounded below by that of g.

When set representations are used, we can use an expression like $f(n) \in \Theta(g(n))$ to mean that f has the same order as g. The set representations also give some nice relationships. For example, we have the following relationships, where the subset relation is proper.

$$O(g(n)) \supset \Theta(g(n)) \cup o(g(n)),$$
$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n)).$$

Exercises

Calculations

1. Find a place to insert the function $\log \log n$ in the sequence (5.40).

2. For each each of the following functions f, find an appropriate place in the sequence (5.40).

   a. $f(n) = \log 1 + \log 2 + \log 3 + \cdots + \log n$.

   b. $f(n) = \log 1 + \log 2 + \log 4 + \cdots + \log 2^n$.

3. For each of the following values of n, calculate the following three numbers: the exact value of $n!$, Stirling's approximation (5.37) for the value of $n!$, and the difference between the two values.

   a. $n = 5$. b. $n = 10$.

Proofs and Challenges

4. Find an example of an increasing function f such that $f(n) = \Theta(1)$.

5. Prove that the binary relation on functions defined by $f(n) = \Theta(g(n))$ is an equivalence relation.

6. For any constant $k > 0$, prove each of the following statements.

   a. $\log(kn) = \Theta(\log n)$.
   b. $\log(k + n) = \Theta(\log n)$.

7. Prove the following sequence of orders: $n \prec n \log n \prec n^2$.

8. For any constant k, show that n^k has lower order than 2^n.

9. Prove the following sequence of orders: $2^n \prec n! \prec n^n$.

10. Let $f(n) = O(h(n))$ and $g(n) = O(h(n))$. Prove each of the following statements.

    a. $af(n) = O(h(n))$ for any real number a.
    b. $f(n) + g(n) = O(h(n))$.

11. Show that each of the following subset relations is proper.

    a. $O(g(n)) \supset \Theta(g(n)) \cup o(g(n))$.
    b. $o(g(n)) \subset O(g(n)) - \Theta(g(n))$.

    Hint: For example, let $g(n) = n$ and let $f(n) =$ if n is odd then 1 else n. Then show that $f(n) \in O(g(n)) - \Theta(g(n)) \cup o(g(n))$ for part (a) and show that $f(n) \in (O(g(n)) - \Theta(g(n))) - o(g(n))$ for part (b).

5.6 Chapter Summary

This chapter introduces some basic tools and techniques that are used to analyze algorithms. Analysis by worst-case running time is discussed. A lower bound is a value that can't be beat by any algorithm in a particular class. An algorithm is optimal if its performance matches the lower bound.

Counting problems often give rise to finite sums that need closed form solutions. Properties of sums together with summation notation provide us with techniques to find closed forms for many finite sums.

Two useful things to count are permutations, in which order is important, and combinations, in which order is not important. Pascal's triangle contains formulas for combinations, which are the same as binomial coefficients. There are formulas to count permutations and combinations of bags; these allow repeated elements. Discrete probability—with finite sample spaces—gives us the tools to define the average-case performance of an algorithm.

Counting problems often give rise to recurrences. Some simple recurrences can be solved by either substitution or cancellation to obtain a finite sum, which can be then transformed into a closed form. The use of generating functions provides a powerful technique for solving recurrences.

Often it makes sense to find approximations for functions that describe the number of operations performed by an algorithm. The rates of growth of two functions can be compared in various ways—big theta, little oh, big oh, and big omega.

Notes

In this chapter we've just scratched the surface of techniques for manipulating expressions that crop up in counting things while analyzing algorithms. The book by Knuth [1968] contains the first account of a collection of techniques for the analysis of algorithms. The book by Graham, Knuth, and Patashnik [1989] contains a host of techniques, formulas, anecdotes, and further references to the literature. The book also introduces an alternative notation for working with sums, which often makes it easier to manipulate them without having to change the expressions for the upper and lower limits of summation. The notation is called Iverson's convention, and it is also described in the article by Knuth [1992].

chapter 6

Elementary Logic

... if it was so, it might be; and if it were so, it would be: but as it isn't, it ain't. That's logic.

—Tweedledee in *Through the Looking-Glass* by Lewis Carroll (1832–1898)

Why is it important to study logic? Two things that we continually try to accomplish are to understand and to be understood. We attempt to understand an argument given by someone so that we can agree with the conclusion or, possibly, so that we can say that the reasoning does not make sense. We also attempt to express arguments to others without making a mistake. A formal study of logic will help improve these fundamental communication skills.

Why should a student of computer science study logic? A computer scientist needs logical skills to argue whether a problem can be solved on a machine, to transform logical statements from everyday language to a variety of computer languages, to argue that a program is correct, and to argue that a program is efficient. Computers are constructed from logic devices and are programmed in a logical fashion. Computer scientists must be able to understand and apply new ideas and techniques for programming, many of which require a knowledge of the formal aspects of logic.

In this chapter we'll discuss the formal character of sentences that contain words like "and," "or," and "not" or a phrase like "if A then B."

chapter guide

Section 6.1 starts our study of logic with the philosophical question "How do we reason?" We'll discuss some common things that we all do when we reason, and we'll introduce the general idea of a "calculus" as a thing with which to study logic.

Section 6.2 introduces the basic notions and notations of propositional calculus. We'll discuss the properties of tautology, contradiction, and contingency.

We'll introduce the idea of equivalence, and we'll use it to find disjunctive and conjunctive normal forms for formulas.

Section 6.3 introduces basic techniques of formal reasoning. We'll introduce rules of inference that will allow us to write proofs in a formal manner that still reflect the way we do informal reasoning.

Section 6.4 introduces axiomatic systems. We'll look at a specific set of three axioms and a single rule of inference that are sufficient to prove any required statement of the propositional calculus.

6.1 How Do We Reason?

How do we reason with each other in our daily lives? We probably state some facts and then state a conclusion based on the facts. For example, the words and the phrase in the following list are often used to indicate that some kind of conclusion is being made:

therefore, thus, whence, so, ergo, hence, it follows that.

When we state a conclusion of some kind, we are applying a rule of logic called an *inference rule.*

The most common rule of inference is called *modus ponens,* and it works like this: Suppose A and B are two statements and we assume that A and "If A then B" are both true. We can then infer that B is true. A typical example of inference by modus ponens is given by the following three sentences:

If it is raining, then there are clouds in the sky.

It is raining.

Therefore, there are clouds in the sky.

We use the modus ponens inference rule without thinking about it. We certainly learned it when we were children, probably by testing a parent. For example, if a child receives a hug from a parent after performing some action, it might dawn on the child that the hug follows after the action. The parent might reinforce the situation by saying, "If you do that again, then you will be rewarded." Parents often make statements such as: "If you touch that stove burner, then you will burn your finger." After touching the burner, the child probably knows a little bit more about modus ponens. A parent might say, "If you do that again, then you are going to be punished." The normal child probably will do it again and notice that punishment follows. Eventually, in the child's mind, the statement "If... then... punishment" is accepted as a true statement, and the modus ponens rule has taken root.

Another inference rule, which we also learned when we were children, is called *modus tollens*, and it works like this: Suppose A and B are any two statements. If the statement "If A then B" is true and the statement B is false, then we infer the falsity of statement A. A child might learn this rule initially by thinking, "If I'm not being punished, then I must not be doing anything wrong."

Most of us are also familiar with the false reasoning exhibited by the *non sequitur*, which means "It does not follow." For example, someone might make several true statements and then conclude that some other statement is true, even though it has nothing to do with the assumptions. The hope is that we can recognize this kind of false reasoning so that we never use it. For example, the following three sentences form a non sequitur:

Io is a moon of Jupiter.
Titan is a moon of Saturn.
Therefore, Earth is the third planet from the sun.

Here's another example of a non sequitur:

You squandered the money entrusted to you.
You did not keep required records.
You incurred more debt than your department is worth.
Therefore, you deserve a promotion.

So we reason by applying inference rules to sentences that we assume are true, obtaining new sentences that we conclude are true. Each of us has our own personal reasoning system in which the assumptions are those English sentences that we assume are true and the inference rules are all the rules that we personally use to convince other people that something is true. But there's a problem.

When two people disagree on what they assume to be true or on how they reason about things, then they have problems trying to reason with each other. Some people call this "lack of communication." Other people call it something worse, especially when things like non sequiturs are part of a person's reasoning system. Can common ground be found? Are there any reasoning systems that are, or should be, contained in everyone's personal reasoning system? The answer is yes. The study of logic helps us understand and describe the fundamental parts of all reasoning systems.

6.1.1 What Is a Calculus?

The Romans used small beads called "calculi" to perform counting tasks. The word "calculi" is the plural of the word "calculus." So it makes sense to think that "calculus" has something to do with calculating. Since there are many kinds of calculation, it shouldn't surprise us that "calculus" is used in many different contexts. Let's give a definition.

A *calculus* is a language of expressions of some kind, with definite rules for forming the expressions. There are values, or meanings, associated with the expressions, and there are definite rules to transform one expression into another expression having the same value.

The English language is something like a calculus, where the expressions are sentences formed by English grammar rules. Certainly, we associate meanings with English sentences. But there are no definite rules for transforming one sentence into another. So our definition of a calculus is not quite satisfied. Let's try again with a programming language X. We'll let the expressions be the programs written in the X language. Is this a calculus? Well, there are certainly rules for forming the expressions, and the expressions certainly have meaning. Are there definite rules for transforming one X language program into another X language program? For most modern programming languages the answer is no. So we don't quite have a calculus. We should note that compilers transform X language programs into Y language programs, where X and Y are different languages. Thus a compiler does not qualify as a calculus transformation rule.

In mathematics the word "calculus" usually means the calculus of real functions. For example, the two expressions

$$D_x[f(x)g(x)] \quad \text{and} \quad f(x)D_xg(x) + g(x)D_xf(x)$$

are equivalent in this calculus. The calculus of real functions satisfies our definition of a calculus because there are definite rules for forming the expressions and there are definite rules for transforming expressions into equivalent expressions.

We'll be studying some different kinds of "logical" calculi. In a logical calculus the expressions are defined by rules, the values of the expressions are related to the concepts of true and false, and there are rules for transforming one expression into another. We'll start with a question.

6.1.2 How Can We Tell Whether Something Is a Proof?

When we reason with each other, we normally use informal proof techniques from our personal reasoning systems. This brings up a few questions:

What is an informal proof?
What is necessary to call something a proof?
How can I tell whether an informal proof is correct?
Is there a proof system to learn for each subject of discussion?
Can I live my life without all this?

A formal study of logic will provide us with some answers to these questions. We'll find general methods for reasoning that can be applied informally in many different situations. We'll introduce a precise language for expressing arguments formally, and we'll discuss ways to translate an informal argument into a formal argument. This is especially important in computer science, in which formal solutions (programs) are required for informally stated problems.

6.2 Propositional Calculus

To discuss reasoning, we need to agree on some rules and notation about the truth of sentences. A sentence that is either true or false is called a *proposition*. For example, each of the following lines contains a proposition:

P	Q	$\neg P$	$P \vee Q$	$P \wedge Q$	$P \to Q$
true	true	false	true	true	true
true	false	false	true	false	false
false	true	true	true	false	true
false	false	true	false	false	true

Figure 6.1 Truth tables.

Winter begins in June in the Southern Hemisphere.
$2 + 2 = 4$.
If it is raining, then there are clouds in the sky.
I may or may not go to a movie tonight.
All integers are even.
There is a prime number greater than a googol.

For this discussion we'll denote propositions by the letters P, Q, and R, possibly subscripted. Propositions can be combined to form more complicated propositions, just the way we combine sentences, using the words "not," "and," "or," and the phrase "if... then...". These combining operations are often called *connectives.* We'll denote them by the following symbols and words:

$\neg$	not, negation.
$\wedge$	and, conjunction.
$\vee$	or, disjunction.
$\to$	conditional, implication.

Some common ways to read the expression $P \to Q$ are "if P then Q," "Q if P," "P implies Q," "P is a sufficient condition for Q," and "Q is a necessary condition for P." P is called the *antecedent*, *premise*, or *hypothesis*, and Q is called the *consequent* or *conclusion* of $P \to Q$.

Now that we have some symbols, we can denote propositions in symbolic form. For example, if P denotes the proposition "It is raining" and Q denotes the proposition "There are clouds in the sky," then $P \to Q$ denotes the proposition "If it is raining, then there are clouds in the sky." Similarly, $\neg P$ denotes the proposition "It is not raining."

The four logical operators are defined to reflect their usage in everyday English. Figure 6.1 is a *truth table* that defines the operators for all possible truth values of their operands.

6.2.1 Well-Formed Formulas and Semantics

Like any programming language or any natural language, whenever we deal with symbols, at least two questions always arise. The first deals with syntax: Is an expression grammatically (or syntactically) correct? The second deals with semantics: What is the meaning of an expression? Let's look at the first question first.

A grammatically correct expression is called a *well-formed formula*, or *wff* for short, which can be pronounced "woof." To decide whether an expression is a wff, we need to precisely define the syntax (or grammar) rules for the formation of wffs in our language. So let's do it.

Syntax

As with any language, we must agree on a set of symbols to use as the alphabet. For our discussion we will use the following sets of symbols:

Truth symbols:	true, false
Connectives:	$\neg$, $\rightarrow$, $\wedge$, $\vee$
Propositional variables:	Uppercase letters like P, Q, and R
Punctuation symbols:	(,).

Next we need to define those expressions (strings) that form the wffs of our language. We do this by giving the following informal inductive definition for the set of propositional wffs.

The Definition of a Wff

A wff is either a truth symbol, or a propositional variable, or the negation of a wff, or the conjunction of two wffs, or the disjunction of two wffs, or the implication of one wff from another, or a wff surrounded by parentheses.

For example, the following expressions are wffs:

$$\text{true, false, } P, \neg\, Q, P \wedge Q, P \rightarrow Q, (P \vee Q) \wedge R, P \wedge Q \rightarrow R.$$

If we need to justify that some expression is a wff, we can apply the inductive definition. Let's look at an example.

example 6.1 Analyzing a Wff

We'll show that the expression $P \wedge Q \vee R$ is a wff. First, we know that P, Q, and R are wffs because they are propositional variables. Therefore, $Q \vee R$ is a wff because it's a disjunction of two wffs. It follows that $P \wedge Q \vee R$ is a wff because it's a conjunction of two wffs. We could have arrived at the same conclusion by saying that $P \wedge Q$ is a wff and then stating that $P \wedge Q \vee R$ is a wff, since it is the disjunction of two wffs.

end example

Semantics

Can we associate a truth table with each wff? Yes we can, once we agree on a hierarchy of precedence among the connectives. For example, $P \wedge Q \vee R$ is a perfectly good wff. But to find a truth table, we need to agree on which connective to evaluate first. We will define the following hierarchy of evaluation for the connectives of the propositional calculus:

$\neg$	(highest, do first)
$\wedge$	
$\vee$	
$\rightarrow$	(lowest, do last)

We also agree that the operations $\wedge$, $\vee$, and $\rightarrow$ are left associative. In other words, if the same operation occurs two or more times in succession, without parentheses, then evaluate the operations from left to right. Be sure you can tell the reason for each of the following lines, where each line contains a wff together with a parenthesized wff with the same meaning:

$P \vee Q \wedge R$	means	$P \vee (Q \wedge R)$.
$P \rightarrow Q \rightarrow R$	means	$(P \rightarrow Q) \rightarrow R$.
$\neg P \vee Q$	means	$(\neg P) \vee Q$.
$\neg P \rightarrow P \wedge Q \vee R$	means	$(\neg P) \rightarrow ((P \wedge Q) \vee R)$.
$\neg \neg P$	means	$\neg (\neg P)$.

Any wff has a natural syntax tree that clearly displays the hierarchy of the connectives. For example, the syntax tree for the wff $P \wedge (Q \vee \neg R)$ is given by the diagram in Figure 6.2.

Now we can say that any wff has a unique truth table. For example, suppose we want to find the truth table for the wff

$$\neg P \rightarrow Q \wedge R.$$

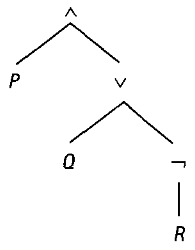


Figure 6.2 Syntax tree.

P	Q	R	$\neg P$	$Q \wedge R$	$\neg P \to Q \wedge R$
true	true	true	false	true	true
true	true	false	false	false	true
true	false	true	false	false	true
true	false	false	false	false	true
false	true	true	true	true	true
false	true	false	true	false	false
false	false	true	true	false	false
false	false	false	true	false	false

Figure 6.3 Truth table.

From the hierarchy of evaluation we know that this wff has the following parenthesized form:

$$(\neg P) \to (Q \wedge R).$$

So we can construct the truth table as follows: Begin by writing down all possible truth values for the three variables P, Q, and R. This gives us a table with eight lines. Next, compute a column of values for $\neg P$. Then compute a column of values for $Q \wedge R$. Finally, use these two columns to compute the column of values for $\neg P \to Q \wedge R$. Figure 6.3 gives the result.

Although we've talked some about meaning, we haven't specifically defined the *meaning*, or *semantics*, of a wff. Let's do it now. We know that any wff has a unique truth table. So we'll associate each wff with its truth table.

The Meaning of a Wff

The meanings of the truth symbols true and false are true and false, respectively. Otherwise, the meaning of a wff is its truth table.

Tautology, Contradiction, and Contingency

A wff is said to be *tautology* if all the truth table values for the wff are true. For example, the wffs $P \vee \neg P$ and $P \to P$ are tautologies. If all the truth table values for a wff are false, then the wff is called a *contradiction*. For example, the wff $P \wedge \neg P$ is a contradiction. If the truth table for a wff contains a true value and a false value, then the wff is called a *contingency*. For example, the wff P is a contingency.

Notational Convenience

We will often use uppercase letters to refer to arbitrary propositional wffs. For example, if we say, "A is a wff," we mean that A represents some arbitrary wff. We also use uppercase letters to denote specific propositional wffs. For example, if we want to talk about the wff $P \wedge (Q \vee \neg R)$ several times in a discussion, we might let $W = P \wedge (Q \vee \neg R)$. Then we can refer to W instead of always writing down the symbols $P \wedge (Q \vee \neg R)$.

6.2.2 Equivalence

In our normal discourse we often try to understand a sentence by rephrasing it in some way. Of course, we always want to make sure that the two sentences have the same meaning. This idea carries over to formal logic too, where we want to describe the idea of equivalence between two wffs.

Definition of Equivalence

Two wffs A and B are *equivalent* (or *logically equivalent*) if they have the same truth value for each assignment of truth values to the set of all propositional variables occurring in the wffs. In this case we write

$$A \equiv B.$$

If two wffs contain the same propositional variables, then they will be equivalent if and only if they have the same truth tables. For example, the wffs $\neg P \vee Q$ and $P \rightarrow Q$ both contain the propositional variables P and Q. The truth tables for the two wffs are shown in Figure 6.4. Since the tables are the same, we have $\neg P \vee Q \equiv P \rightarrow Q$.

Two wffs that do not share the same propositional variables can still be equivalent. For example, the wffs $\neg P$ and $\neg P \vee (Q \wedge \neg P)$ don't share Q. Since the truth table for $\neg P$ has two lines and the truth table for $\neg P \vee (Q \wedge \neg P)$ has four lines, the two truth tables can't be the same. But we can still compare the truth values of the wffs for each truth assignment to the variables occuring in both wffs. We can do this with a truth table using the variables P and Q as shown in Figure 6.5. Since the columns agree, we know the the wffs are equivalent. So we have $\neg P \equiv \neg P \vee (Q \wedge \neg P)$.

When two wffs don't have any propositional variables in common, the only way for them to be equivalent is that they are either both tautologies or both contraditions. Can you see why? For example, $P \vee \neg P \equiv Q \rightarrow Q \equiv$ true.

The definition of equivalence also allows us to make the following useful formulation in terms of conditionals and tautologies.

P	Q	$\neg P \vee Q$	$P \rightarrow Q$
true	true	true	true
true	false	false	false
false	true	true	true
false	false	true	true

Figure 6.4 Equivalent wffs.

P	Q	$\neg P$	$\neg P \vee (Q \wedge \neg P)$
true	true	false	false
true	false	false	false
false	true	true	true
false	false	true	true

Figure 6.5 Equivalent wffs.

Basic Equivalences (6.1)

Negation	*Disjunction*	*Conjunction*	*Implication*
$\neg\neg A \equiv A$	$A \vee$ true $\equiv$ true	$A \wedge$ true $\equiv A$	$A \rightarrow$ true $\equiv$ true
	$A \vee$ false $\equiv A$	$A \wedge$ false $\equiv$ false	$A \rightarrow$ false $\equiv \neg A$
	$A \vee A \equiv A$	$A \wedge A \equiv A$	true $\rightarrow A \equiv A$
	$A \vee \neg A \equiv$ true	$A \wedge \neg A \equiv$ false	false $\rightarrow A \equiv$ true
			$A \rightarrow A \equiv$ true

Some Conversions	*Absorption laws*
$A \rightarrow B \equiv \neg A \vee B$	$A \wedge (A \vee B) \equiv A$
$\neg (A \rightarrow B) \equiv A \wedge \neg B$	$A \vee (A \wedge B) \equiv A$
$A \rightarrow B \equiv A \wedge \neg B \rightarrow$ false	$A \wedge (\neg A \vee B) \equiv A \wedge B$
$\wedge$ and $\vee$ are associative.	$A \vee (\neg A \wedge B) \equiv A \vee B$
$\wedge$ and $\vee$ are commutative.	
$\wedge$ and $\vee$ distribute over each other:	*De Morgan's Laws*
$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$	$\neg (A \wedge B) \equiv \neg A \vee \neg B$
$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$	$\neg (A \vee B) \equiv \neg A \wedge \neg B$

Figure 6.6 Equivalences, conversions, basic laws.

Equivalence

$A \equiv B$ if and only if $(A \rightarrow B) \wedge (B \rightarrow A)$ is a tautology
if and only if $A \rightarrow B$ and $B \rightarrow A$ are tautologies.

Before we go much further, let's list a few basic equivalences. Figure 6.6 shows a collection of equivalences, all of which are easily verified by truth tables, so we'll leave them as exercises.

Reasoning with Equivalences

Can we do anything with the basic equivalences? Sure. We can use them to show that other wffs are equivalent without checking truth tables. But first we need to observe two general properties of equivalence.

The first thing to observe is that equivalence is an "equivalence" relation. In other words, $\equiv$ satisfies the reflexive, symmetric, and transitive properties. The transitive property is the most important property for our purposes. It can be stated as follows for any wffs W, X, and Y:

$$\text{If } W \equiv X \text{ and } X \equiv Y, \text{ then } W \equiv Y.$$

This property allows us to write a sequence of equivalences and then conclude that the first wff is equivalent to the last wff, just the way we do it with ordinary equality of algebraic expressions.

The next thing to observe is the *replacement rule* of equivalences, which is similar to old rule: "Substituting equals for equals doesn't change the value of an expression."

Replacement Rule

Any subwff (i.e., a wff that is part of another wff) can be replaced by an equivalent wff without changing the truth value of the original wff.

Can you see why this is OK for equivalences? For example, suppose we want to simplify the wff $B \rightarrow (A \vee (A \wedge B))$. We might notice that one of the laws from (6.1) gives $A \vee (A \wedge B) \equiv A$. Therefore, we can apply the replacement rule and write the following equivalence:

$$B \rightarrow (A \vee (A \wedge B)) \equiv B \rightarrow A.$$

Let's do an example to illustrate the process of showing that two wffs are equivalent without checking truth tables.

example

6.2 A Conditional Relationship

The following equivalence shows an interesting relationship involving the connective $\rightarrow$.

$$A \rightarrow (B \rightarrow C) \equiv B \rightarrow (A \rightarrow C).$$

We'll prove it using equivalences that we already know. Make sure you can give the reason for each line of the proof.

$$\begin{aligned}
\text{Proof:}\quad A \rightarrow (B \rightarrow C) &\equiv A \rightarrow (\neg B \vee C) \\
&\equiv \neg A \vee (\neg B \vee C) \\
&\equiv (\neg A \vee \neg B) \vee C \\
&\equiv (\neg B \vee \neg A) \vee C \\
&\equiv \neg B \vee (\neg A \vee C) \\
&\equiv B \rightarrow (\neg A \vee C) \\
&\equiv B \rightarrow (A \rightarrow C). \text{ QED}
\end{aligned}$$

end example

This example illustrates that we can use known equivalences like (6.1) as rules to transform wffs into other wffs that have the same meaning. This justifies the word "calculus" in the name "propositional calculus."

Is It a Tautology, a Contradiction, or a Contingency?

Suppose our task is to find whether a wff W is a tautology, a contradiction, or a contingency. If W contains n variables, then there are 2^n different assignments of truth values to the variables of W. Building a truth table with 2^n rows can be tedious when n is moderately large.

Are there any other ways to determine the meaning of a wff? Yes. One way is to use equivalences to transform the wff into a wff that we recognize as a tautology, a contradition, or a contingency. But another way, called Quine's method, combines the substitution of variables with the use of equivalences. To describe the method we need a definition.

Definition

If A is a variable in the wff W, then the expression $W(A/\text{true})$ denotes the wff obtained from W by replacing all occurrences of A by true. Similarly, we define $W(A/\text{false})$ to be the wff obtained from W by replacing all occurrences of A by false.

For example, if $W = (A \to B) \wedge (A \to C)$, then $W(A/\text{true})$ and $W(A/\text{false})$ have the following values, where we've continued in each case with some basic equivalences.

$$W(A/\text{true}) = (\text{true} \to B) \wedge (\text{true} \to C) \equiv B \wedge C.$$
$$W(A/\text{false}) = (\text{false} \to B) \wedge (\text{false} \to C) \equiv \text{true} \wedge \text{true} \equiv \text{true}.$$

Now comes the key observation that allows us to use these ideas to decide the truth value of a wff.

Substitution Properties

1. W is a tautology iff $W(A/\text{true})$ and $W(A/\text{false})$ are tautologies.
2. W is a contradiction iff $W(A/\text{true})$ and $W(A/\text{false})$ are contradictions.

For example, in our little example we found that $W(A/\text{true}) \equiv B \wedge C$, which is a contingency, and $W(A/\text{false}) \equiv \text{true}$, which is a tautology. Therefore, W is a contingency.

The idea of Quine's method is to construct $W(A/\text{true})$ and $W(A/\text{false})$ and then to simplify these wffs by using the basic equivalences. If we can't tell the truth values, then choose another variable and apply the method to each of these wffs. A complete example is in order.

example **6.3 Quine's Method**

Suppose that we want to check the meaning of the following wff W:

$$[(A \wedge B \rightarrow C) \wedge (A \rightarrow B)] \rightarrow (A \rightarrow C).$$

First we compute the two wffs $W(A/\text{true})$ and $W(A/\text{false})$ and simplify them.

$$\begin{aligned} W(A/\text{true}) &= [(\text{true} \wedge B \rightarrow C) \wedge (\text{true} \rightarrow B)] \rightarrow (\text{true} \rightarrow C) \\ &\equiv [(B \rightarrow C) \wedge (\text{true} \rightarrow B)] \rightarrow (\text{true} \rightarrow C) \\ &\equiv [(B \rightarrow C) \wedge B] \rightarrow C. \end{aligned}$$

$$\begin{aligned} W(A/\text{false}) &= [(\text{false} \wedge B \rightarrow C) \wedge (\text{false} \rightarrow B)] \rightarrow (\text{false} \rightarrow C) \\ &\equiv [(\text{false} \rightarrow C) \wedge \text{true}] \rightarrow \text{true} \\ &\equiv \text{true}. \end{aligned}$$

Therefore, $W(A/\text{false})$ is a tautology. Now we need to check the simplification of $W(A/\text{true})$. Call it X. We continue the process by constructing the two wffs $X(B/\text{true})$ and $X(B/\text{false})$:

$$\begin{aligned} X(B/\text{true}) &= [(\text{true} \rightarrow C) \wedge \text{true}] \rightarrow C \\ &\equiv [C \wedge \text{true}] \rightarrow C \\ &\equiv C \rightarrow C \\ &\equiv \text{true}. \end{aligned}$$

So $X(B/\text{true})$ is a tautology. Now let's look at $X(B/\text{false})$.

$$\begin{aligned} X(B/\text{false}) &= [(\text{false} \rightarrow C) \wedge \text{false}] \rightarrow C \\ &\equiv [\text{true} \wedge \text{false}] \rightarrow C \\ &\equiv \text{false} \rightarrow C \\ &\equiv \text{true}. \end{aligned}$$

So $X(B/\text{false})$ is also a tautology. Therefore, X is a tautology and it follows that W is a tautology.

end example

Quine's method can also be described graphically with a binary tree. Let W be the root. If N is any node, pick one of its variables, say V, and let the two children of N be $N(V/\text{true})$ and $N(V/\text{false})$. Each node should be simplified as much as possible. Then W is a tautology if all leaves are true, a contradiction if all leaves are false, and a contingency otherwise. Let's illustrate the idea with the wff $P \rightarrow Q \wedge P$. The binary tree in Figure 6.7 shows that the wff $P \rightarrow Q \wedge P$ is a contingency because Quine's method gives one false leaf and two true leaves.

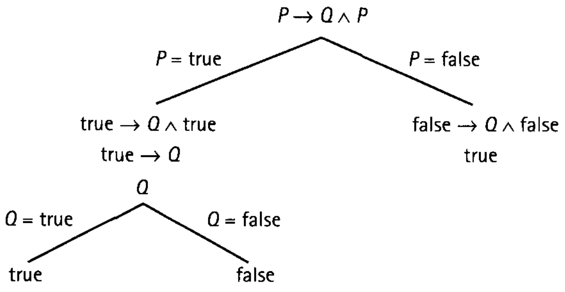

$P \rightarrow Q \wedge P$
P = true
P = false
true $\rightarrow Q \wedge$ true
true $\rightarrow Q$
Q
false $\rightarrow Q \wedge$ false
true
Q = true
Q = false
true
false

Figure 6.7 Quine's method.

6.2.3 Truth Functions and Normal Forms

A *truth function* is a function whose arguments can take only the values true or false and whose values are either true or false. So any wff defines a truth function. For example, the function g defined by

$$g(P, Q) = P \wedge Q$$

is a truth function. Is the converse true? In other words, is every truth function a wff? The answer is yes. To see why this is true, we'll present a technique to construct a wff for any truth function.

For example, suppose we define a truth function f by saying that $f(P, Q)$ is true exactly when P and Q have opposite truth values. Is there a wff that has the same truth table as f? We'll introduce the technique with this example. Figure 6.8 is the truth table for f. We'll explain the statements on the right side of this table.

We've written the two wffs $P \wedge \neg Q$ and $\neg P \wedge Q$ on the second and third lines of the table because the values of f are true on these lines. Each wff is a conjunction of argument variables or their negations according to their values on the same line, according to the following two rules:

If P is true, then put P in the conjunction.
If P is false, then put $\neg P$ in the conjunction.

Let's see why we want to follow these rules. Notice that the truth table for $P \wedge \neg Q$ in Figure 6.9 has exactly one true value and it occurs on the second

P	Q	$f(P, Q)$	
true	true	false	
true	false	true	Create $P \wedge \neg Q$.
false	true	true	Create $\neg P \wedge Q$.
false	false	false	

Figure 6.8 A truth function.

P	Q	$f(P, Q)$	$P \wedge \neg Q$	$\neg P \wedge Q$
true	true	false	false	false
true	false	true	true	false
false	true	true	false	true
false	false	false	false	false

Figure 6.9 A truth function.

line. Similarly, the truth table for $\neg P \wedge Q$ has exactly one true value and it occurs on the third line of the table.

Thus each of the tables for $P \wedge \neg Q$ and $\neg P \wedge Q$ has exactly one true value per column, and these true values occur on the same lines as the true values for f. Since there is one conjunctive wff for each occurrence of true in the table for f, it follows that the table for f can be obtained by taking the disjunction of the tables for $P \wedge \neg Q$ and $\neg P \wedge Q$. Thus we obtain the following equivalence.

$$f(P, Q) \equiv (P \wedge \neg Q) \vee (\neg P \wedge Q).$$

Let's do another example to get the idea. Then we'll discuss the special forms that we obtain by using this technique.

example 6.4 Converting a Truth Function

Let f be the truth function defined as follows:

$$f(P, Q, R) = \text{true if and only if either } P = Q = \text{false or } Q = R = \text{true}.$$

Then f is true in exactly the following four cases:

f(false, false, true),
f(false, false, false),
f(true, true, true),
f(false, true, true).

So we can construct a wff equivalent to f by taking the disjunction of the four wffs that correspond to these four cases. The disjunction follows.

$$(\neg P \wedge \neg Q \wedge R) \vee (\neg P \wedge \neg Q \wedge \neg R) \vee (P \wedge Q \wedge R) \vee (\neg P \wedge Q \wedge R).$$

end example

The method we have described can be generalized to construct an equivalent wff for any truth function having at least one true value. If a truth function doesn't have any true values, then it is a contradiction and is equivalent to false.

So every truth function is equivalent to some propositional wff. We'll state this as the following theorem:

Truth Functions **(6.2)**
Every truth function is equivalent to a propositional wff.

Now we're going to discuss some useful forms for propositional wffs. But first we need a little terminology. A *literal* is a propositional variable or its negation. For example, P, Q, $\neg P$, and $\neg Q$ are literals.

Disjunctive Normal Form

A *fundamental conjunction* is either a literal or a conjunction of two or more literals. For example, P and $P \wedge \neg Q$ are fundamental conjunctions. A *disjunctive normal form* (DNF) is either a fundamental conjunction or a disjunction of two or more fundamental conjunctions. For example, the following wffs are DNFs:

$$P \vee (\neg P \wedge Q),$$
$$(P \wedge Q) \vee (\neg Q \wedge P),$$
$$(P \wedge Q \wedge R) \vee (\neg P \wedge Q \wedge R).$$

Sometimes the trivial cases are hardest to see. For example, try to explain why the following four wffs are DNFs: P, $\neg P$, $P \vee \neg P$, and $\neg P \wedge Q$. The propositions that we constructed for truth functions are DNFs.

It is often the case that a DNF is equivalent to a simpler DNF. For example, the DNF $P \vee (P \wedge Q)$ is equivalent to the simpler DNF P by using (6.1). For another example, consider the following DNF:

$$(P \wedge Q \wedge R) \vee (\neg P \wedge Q \wedge R) \vee (P \wedge R).$$

The first fundamental conjunction is equivalent to $(P \wedge R) \wedge Q$, which we see contains the third fundamental conjunction $P \wedge R$ as a subexpression. Thus the first term of the DNF can be absorbed by (6.1) into the third term, which gives the following simpler equivalent DNF:

$$(\neg P \wedge Q \wedge R) \vee (P \wedge R).$$

For any wff W we can always construct an equivalent DNF. If W is a contradiction, then it is equivalent to the single term DNF $P \wedge \neg P$. If W is not a contradiction, then we can write down its truth table and use the technique that we used for truth functions to construct a DNF. So we can make the following statement.

Disjunctive Normal Form **(6.3)**
Every wff is equivalent to a DNF.

Another way to construct a DNF for a wff is to transform it into a DNF by using the equivalences of (6.1). In fact we'll outline a short method that will always do the job:

First, remove all occurrences (if there are any) of the connective $\rightarrow$ by using the equivalence

$$A \rightarrow B \equiv \neg A \vee B.$$

Next, move all negations inside to create literals by using De Morgan's equivalences

$$\neg (A \wedge B) \equiv \neg A \vee \neg B \text{ and } \neg (A \vee B) \equiv \neg A \wedge \neg B.$$

Finally, apply the distributive equivalences to obtain a DNF. Let's look at an example.

example 6.5 A DNF Construction

We'll construct a DNF for the wff $((P \wedge Q) \rightarrow R) \wedge S$.

$$\begin{aligned}((P \wedge Q) \rightarrow R) \wedge S &\equiv (\neg (P \wedge Q) \vee R) \wedge S \\ &\equiv (\neg P \vee \neg Q \vee R) \wedge S \\ &\equiv (\neg P \wedge S) \vee (\neg Q \wedge S) \vee (R \wedge S).\end{aligned}$$

end example

Suppose W is a wff having n distinct propositional variables. A DNF for W is called a *full disjunctive normal form* if each fundamental conjunction has exactly n literals, one for each of the n variables appearing in W. For example, the following wff is a full DNF:

$$(P \wedge Q \wedge R) \vee (\neg P \wedge Q \wedge R).$$

The wff $P \vee (\neg P \wedge Q)$ is a DNF but not a full DNF because the variable Q does not occur in the first fundamental conjunction.

The truth table technique to construct a DNF for a truth function automatically builds a full DNF because all of the variables in a wff occur in each fundamental conjunction. So we can state the following result.

Full Disjunctive Normal Form **(6.4)**

Every wff that is not a contradiction is equivalent to a full DNF.

Conjunctive Normal Form

In a manner entirely analogous to the previous discussion we can define a *fundamental disjunction* to be either a literal or the disjunction of two or more literals. A *conjunctive normal form* (CNF) is either a fundamental disjunction or a conjunction of two or more fundamental disjunctions. For example, the following wffs are CNFs:

$$P \wedge (\neg P \vee Q),$$
$$(P \vee Q) \wedge (\neg Q \vee P),$$
$$(P \vee Q \vee R) \wedge (\neg P \vee Q \vee R).$$

Let's look at some trivial examples. Notice that the following four wffs are CNFs: P, $\neg P$, $P \wedge \neg P$, and $\neg P \vee Q$. As in the case for DNFs, some CNFs are equivalent to simpler CNFs. For example, the CNF $P \wedge (P \vee Q)$ is equivalent to the simpler CNF P by (6.1).

Suppose some wff W has n distinct propositional letters. A CNF for W is called a *full conjunctive normal form* if each fundamental disjunction has exactly n literals, one for each of the n variables that appear in W. For example, the following wff is a full CNF:

$$(P \vee Q \vee R) \wedge (\neg P \vee Q \vee R).$$

On the other hand, the wff $P \wedge (\neg P \vee Q)$ is a CNF but not a full CNF.

It's possible to write any truth function f that is not a tautology as a full CNF. In this case we associate a fundamental disjunction with each line of the truth table in which f has a false value, with the property that the fundamental disjunction is false on only that line. Let's return to our original example, in which $f(P, Q)$ is true exactly when P and Q have opposite truth values. Figure 6.10 shows the truth values for f along with a fundamental disjunction created for each line with a false value.

In this case, $\neg P$ is added to the disjunction if P = true, and P is added to the disjunction if P = false. Then we take the conjunction of these disjunctions to obtain the following conjunctive normal form of f:

$$f(P, Q) \equiv (\neg P \vee \neg Q) \wedge (P \vee Q).$$

P	Q	$f(P, Q)$	
true	true	false	Create $\neg P \vee \neg Q$.
true	false	true	
false	true	true	
false	false	false	Create $P \vee Q$.

Figure 6.10 A truth function.

Of course, any tautology is equivalent to the single term CNF $P \vee \neg P$. Now we can state the following results for CNFs, which correspond to statements (6.3) and (6.4) for DNFs:

Conjunctive Normal Form

Every wff is equivalent to a CNF. (6.5)

Every wff that is not a tautology is equivalent to a full CNF. (6.6)

We should note that some authors use the terms "disjunctive normal form" and "conjunctive normal form" to describe the expressions that we have called "full disjunctive normal forms" and "full conjunctive normal forms." For example, they do not consider $P \vee (\neg P \wedge Q)$ to be a DNF. We use the more general definitions of DNF and CNF because they are useful in describing methods for automatic reasoning and they are useful in describing methods for simplifying digital logic circuits.

Constructing Full Normal Forms using Equivalences

We can construct full normal forms for wffs without resorting to truth table techniques. Let's start with the full disjunctive normal form. To find a full DNF for a wff, we first convert it to a DNF by the usual actions: eliminate conditionals, move negations inside, and distribute $\wedge$ over $\vee$. For example, the wff $P \wedge (Q \rightarrow R)$ can be converted to a DNF in two steps, as follows:

$$\begin{aligned} P \wedge (Q \rightarrow R) &\equiv P \wedge (\neg Q \vee R) \\ &\equiv (P \wedge \neg Q) \vee (P \wedge R). \end{aligned}$$

The right side of the equivalence is a DNF. However, it's not a full DNF because the two fundamental conjunctions don't contain all three variables. The trick to add the extra variables can be described as follows:

To add a variable, say R, to a fundamental conjunction C without changing the value of C, write the following equivalences:

$$C \equiv C \wedge \text{true} \equiv C \wedge (R \vee \neg R) \equiv (C \wedge R) \vee (C \wedge \neg R).$$

Let's continue with our example. First, we'll add the letter R to the fundamental conjunction $P \wedge \neg Q$. Be sure to justify each step of the following calculation:

$$\begin{aligned} P \wedge \neg Q &\equiv (P \wedge \neg Q) \wedge \text{true} \\ &\equiv (P \wedge \neg Q) \wedge (R \vee \neg R) \\ &\equiv (P \wedge \neg Q \wedge R) \vee (P \wedge \neg Q \wedge \neg R). \end{aligned}$$

Next, we'll add the variable Q to the fundamental conjunction $P \wedge R$:

$$\begin{aligned} P \wedge R &\equiv (P \wedge R) \wedge \text{true} \\ &\equiv (P \wedge R) \wedge (Q \vee \neg Q) \\ &\equiv (P \wedge R \wedge Q) \vee (P \wedge R \wedge \neg Q). \end{aligned}$$

Lastly, we put the two wffs together to obtain a full DNF for $P \wedge (Q \rightarrow R)$:

$$(P \wedge \neg Q \wedge R) \vee (P \wedge \neg Q \wedge \neg R) \vee (P \wedge R \wedge Q) \vee (P \wedge R \wedge \neg Q).$$

example

6.6 Constructing a Full DNF

We'll construct a full DNF for the wff $P \rightarrow Q$. Make sure to justify each line of the following calculation.

$$\begin{aligned} P \rightarrow Q &\equiv \neg P \vee Q \\ &\equiv (\neg P \wedge \text{true}) \vee Q \\ &\equiv (\neg P \wedge (Q \vee \neg Q)) \vee Q \\ &\equiv (\neg P \wedge Q) \vee (\neg P \wedge \neg Q) \vee Q \\ &\equiv (\neg P \wedge Q) \vee (\neg P \wedge \neg Q) \vee (Q \wedge \text{true}) \\ &\equiv (\neg P \wedge Q) \vee (\neg P \wedge \neg Q) \vee (Q \wedge (P \vee \neg P)) \\ &\equiv (\neg P \wedge Q) \vee (\neg P \wedge \neg Q) \vee (Q \wedge P) \vee (Q \wedge \neg P) \\ &\equiv (\neg P \wedge Q) \vee (\neg P \wedge \neg Q) \vee (Q \wedge P). \end{aligned}$$

end example

We can proceed in an entirely analogous manner to find a full CNF for a wff. The trick in this case is to add variables to a fundamental disjunction without changing its truth value. It goes as follows:

To add a variable, say R, to a fundamental disjunction D without changing the value of D, write the following equivalences:

$$D \equiv D \vee \text{false} \equiv D \vee (R \wedge \neg R) \equiv (D \vee R) \wedge (D \vee \neg R).$$

For example, let's find a full CNF for the wff $P \wedge (P \rightarrow Q)$. To start off, we put the wff in conjunctive normal form as follows:

$$P \wedge (P \rightarrow Q) \equiv P \wedge (\neg P \vee Q).$$

The right side is not a full CNF because the variable Q does not occur in the fundamental disjunction P. So we'll apply the trick to add the variable Q. Make sure you can justify each step in the following calculation:

$$\begin{aligned} P \wedge (P \rightarrow Q) &\equiv P \wedge (\neg P \vee Q) \\ &\equiv (P \vee \text{false}) \wedge (\neg P \vee Q) \\ &\equiv (P \vee (Q \wedge \neg Q)) \wedge (\neg P \vee Q) \\ &\equiv (P \vee Q) \wedge (P \vee \neg Q) \wedge (\neg P \vee Q). \end{aligned}$$

The result is a full CNF that is equivalent to the original wff. Let's do another example.

example **6.7 Constructing a Full CNF**

We'll construct a full CNF for $(P \rightarrow (Q \vee R)) \wedge (P \vee Q)$. After converting the wff to conjunctive normal form, all we need to do is add the variable R to the fundamental disjunction $P \vee Q$. Here's the transformation:

$$\begin{aligned} (P \rightarrow (Q \vee R)) \wedge (P \vee Q) &\equiv (\neg P \vee Q \vee R) \wedge (P \vee Q) \\ &\equiv (\neg P \vee Q \vee R) \wedge ((P \vee Q) \vee \text{false}) \\ &\equiv (\neg P \vee Q \vee R) \wedge ((P \vee Q) \vee (R \wedge \neg R)) \\ &\equiv (\neg P \vee Q \vee R) \wedge ((P \vee Q \vee R) \wedge (P \vee Q \vee \neg R)) \\ &\equiv (\neg P \vee Q \vee R) \wedge (P \vee Q \vee R) \wedge (P \vee Q \vee \neg R). \end{aligned}$$

end example

6.2.4 Complete Sets of Connectives

The four connectives in the set $\{\neg, \wedge, \vee, \rightarrow\}$ are used to form the wffs of the propositional calculus. Are there other sets of connectives that will do the same job? The answer is yes. A set of connectives is called *complete* if every wff of the propositional calculus is equivalent to a wff using connectives from the set. We've already seen that every wff has a disjunctive normal form, which uses only the connectives $\neg$, $\wedge$, and $\vee$. Therefore, $\{\neg, \wedge, \vee\}$ is a complete set of connectives for the propositional calculus. Recall that we don't need implication because we have the equivalence

$$A \rightarrow B \equiv \neg A \vee B.$$

For a second example, consider the two connectives $\neg$ and $\vee$. To show that these connectives generate the propositional calculus, we need only show that statements of the form $A \wedge B$ can be written in terms of $\neg$ and $\vee$. This can be seen by the equivalence

$$A \wedge B \equiv \neg (\neg A \vee \neg B).$$

P	*Q*	NAND (*P*, *Q*)
true	true	false
true	false	true
false	true	true
false	false	true

Figure 6.11 A truth table.

Therefore, $\{\neg, \vee\}$ is a complete set of connectives for the propositional calculus because we know that $\{\neg, \wedge, \vee\}$ is a complete set. Other complete sets are $\{\neg, \wedge\}$ and $\{\neg, \rightarrow\}$. We'll leave these as exercises. Are there any single connectives that are complete? The answer is yes, but we won't find one among the four basic connectives. There is a connective called the NAND operator, which is short for the "Negation of AND." We'll write NAND in functional form NAND(P, Q), since there is no well-established symbol for it. Figure 6.11 shows the truth table for NAND.

To see that NAND is complete, we have to show that the other connectives can be defined in terms of it. For example, we can write negation in terms of NAND as follows:

$$\neg P \equiv \text{NAND}(P, P).$$

We'll leave it as an exercise to show that the other connectives can be written in terms of NAND.

Another single connective that is complete for the propositional calculus is the NOR operator. NOR is short for the "Negation of OR." Figure 6.12 shows the truth table for NOR.

We'll leave it as an exercise to show that NOR is a complete connective for the propositional calculus. NAND and NOR are important because they represent the behavior of two important building blocks for logic circuits.

P	*Q*	NOR (*P*, *Q*)
true	true	false
true	false	false
false	true	false
false	false	true

Figure 6.12 A truth table.

Exercises

Syntax and Symmantics

1. Write down the parenthesized version of each of the following expressions.
   a. $\neg P \wedge Q \rightarrow P \vee R$.
   b. $P \vee \neg Q \wedge R \rightarrow P \vee R \rightarrow \neg Q$.
   c. $A \rightarrow B \vee \neg C \wedge D \wedge E \rightarrow F$.

2. Remove as many parentheses as possible from each of the following wffs.
   a. $(((P \vee Q) \rightarrow (\neg R)) \vee (((\neg Q) \wedge R) \wedge P))$.
   b. $((A \rightarrow (B \vee C)) \rightarrow (A \vee (\neg (\neg B))))$.

3. Let A, B, and C be propositional wffs. Find a wff whose meaning is reflected by the statement "If A then B else C."

Equivalence

4. Use truth tables to verify the equivalences in (6.1).

5. Use other equivalences to prove the equivalence

$$A \rightarrow B \equiv A \wedge \neg B \rightarrow \text{false}.$$

   Hint: Start with the right side.

6. Show that $\rightarrow$ is not associative. That is, show that $(A \rightarrow B) \rightarrow C$ is not equivalent to $A \rightarrow (B \rightarrow C)$.

7. Use Quine's method to show that each wff is a contingency.
   a. $A \vee B \rightarrow B$.
   b. $(A \rightarrow B) \wedge (B \rightarrow \neg A) \rightarrow A$.
   c. $(A \rightarrow B) \wedge (B \rightarrow C) \rightarrow (C \rightarrow A)$.
   d. $(A \vee B \rightarrow C) \wedge A \rightarrow (C \rightarrow B)$.
   e. $(A \rightarrow B) \vee ((C \rightarrow \neg B) \wedge \neg C)$.
   f. $(A \vee B) \rightarrow (C \vee A) \wedge (\neg C \vee B)$.

8. Use Quine's method to show that each wff is a tautology.
   a. $(A \rightarrow B) \wedge (B \rightarrow C) \rightarrow (A \rightarrow C)$.
   b. $(A \vee B) \wedge (A \rightarrow C) \wedge (B \rightarrow D) \rightarrow (C \vee D)$.
   c. $(A \rightarrow C) \wedge (B \rightarrow D) \wedge (\neg C \vee \neg D) \rightarrow (\neg A \vee \neg B)$.
   d. $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$.
   e. $(\neg B \rightarrow \neg A) \rightarrow ((\neg B \rightarrow A) \rightarrow B)$.
   f. $(A \rightarrow B) \rightarrow (C \vee A \rightarrow C \vee B)$.
   g. $(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow (A \vee B \rightarrow C))$.
   h. $(A \rightarrow B) \rightarrow (\neg (B \wedge C) \rightarrow \neg (C \wedge A))$.

9. Verify each of the following equivalences by writing an equivalence proof. That is, start on one side and use known equivalences to get to the other side.
   a. $(A \to B) \wedge (A \vee B) \equiv B$.
   b. $A \wedge B \to C \equiv (A \to C) \vee (B \to C)$.
   c. $A \wedge B \to C \equiv A \to (B \to C)$.
   d. $A \vee B \to C \equiv (A \to C) \wedge (B \to C)$.
   e. $A \to B \wedge C \equiv (A \to B) \wedge (A \to C)$.
   f. $A \to B \vee C \equiv (A \to B) \vee (A \to C)$.

10. Show that each wff is a tautology by using equivalences to show that each wff is equivalent to true.
   a. $A \to A \vee B$.
   b. $A \wedge B \to A$.
   c. $(A \vee B) \wedge \neg A \to B$.
   d. $A \to (B \to A)$.
   e. $(A \to B) \wedge \neg B \to \neg A$.
   f. $(A \to B) \wedge A \to B$.
   g. $A \to (B \to (A \wedge B))$.
   h. $(A \to B) \to ((A \to \neg B) \to \neg A)$.

Normal Forms

11. Use equivalences to transform each of the following wffs into a DNF.
   a. $(P \to Q) \to P$.
   b. $P \to (Q \to P)$.
   c. $Q \wedge \neg P \to P$.
   d. $(P \vee Q) \wedge R$.
   e. $P \to Q \wedge R$.
   f. $(A \vee B) \wedge (C \to D)$.

12. Use equivalences to transform each of the following wffs into a CNF.
   a. $(P \to Q) \to P$.
   b. $P \to (Q \to P)$.
   c. $Q \wedge \neg P \to P$.
   d. $(P \vee Q) \wedge R$.
   e. $P \to Q \wedge R$.
   f. $(A \wedge B) \vee E \vee F$.
   g. $(A \wedge B) \vee (C \wedge D) \vee (E \to F)$.

13. For each of the following functions, write down the full DNF and full CNF representations.

a. $f(P, Q) =$ true if and only if P is true.
b. $f(P, Q, R) =$ true if and only if either Q is true or R is false.

14. Transform each of the following wffs into a full DNF if possible.
    a. $(P \to Q) \to P$.
    b. $Q \wedge \neg P \to P$.
    c. $P \to (Q \to P)$.
    d. $(P \vee Q) \wedge R$.
    e. $P \to Q \wedge R$.

15. Transform each of the following wffs into a full CNF if possible.
    a. $(P \to Q) \to P$.
    b. $P \to (Q \to P)$.
    c. $Q \wedge \neg P \to P$.
    d. $P \to Q \wedge R$.
    e. $(P \vee Q) \wedge R$.

Challenges

16. Show that each of the following sets of operations is a complete set of connectives for the propositional calculus.
    a. $\{\neg, \wedge\}$. b. $\{\neg, \to\}$. c. $\{\text{false}, \to\}$.
    d. $\{\text{NAND}\}$. e. $\{\text{NOR}\}$.

17. Show that there are no complete single binary connectives other than NAND and NOR. *Hint:* Let f be the truth function for a complete binary connective. Show that $f(\text{true, true}) =$ false and $f(\text{false, false}) =$ true because the negation operation must be represented in terms of f. Then consider the remaining cases in the truth table for f.

6.3 Formal Reasoning

We have seen that truth tables are sufficient to find the truth of any proposition. However, if a proposition has three or more variables and contains several connectives, then a truth table can become quite complicated. When we use an equivalence proof rather than truth tables to decide the equivalence of two wffs, it seems a bit closer to the way we communicate with each other.

Although there is no need to formally reason about the truth of propositions, it turns out that all other parts of logic need tools other than truth tables to reason about the truth of wffs. So we'll introduce the basic ideas of a formal reasoning system. We'll do it here because the techniques carry over to all logical systems. A formal reasoning system must have a set of well-formed formulas (wffs) to represent the statements of interest. But two other ingredients are required, and we'll discuss them next.

6.3.1 Inference Rules

A reasoning system needs some rules to help us conclude things. An *inference rule* maps one or more wffs, called *premises*, to a single wff, called the *conclusion*. For example, the modus ponens rule maps the wffs A and $A \to B$ to the wff B. If we let MP stand for modus ponens, then we can represent the rule by writing

$$\text{MP}(A,\ A \to B) = B.$$

In general, if R is an inference rule and $R(P_1,\ldots,\ P_k) = C$, then we say something like, "C is inferred from P_1 and ... and P_k by R." A common way to represent an inference rule is to draw a horizontal line, place the premises above the line, and place the conclusion below the line prefixed by the symbol $\therefore$ as follows.

$$\frac{P_1, \cdots, P_k}{\therefore C}.$$

The symbol $\therefore$ can be read as any of the following words.

therefore, thus, whence, so, ergo, hence.

We can also say, "C is a direct consequence of P_1 and ... and P_k." For example, the modus ponens rule can be written as follows.

Modus Ponens (MP) (6.7)

$$\frac{A \to B, A}{\therefore B}.$$

We would like our inference rules to preserve truth. In other words, if all the premises are tautologies, then we want the conclusion to be a tautology. So an inference rule $R(P_1,\ldots,\ P_k) = C$ preserves truth if the following wff is a tautology.

$$P_1 \wedge \cdots \wedge P_k \to C.$$

We should note that each of the inference rules that we discuss here preserves truth (see the exercises in Section 6.2). For example, the modus ponens rule preserves truth because $A \wedge (A \to B) \to B$ is a tautology.

Any conditional tautology can be used as the basis for an inference rule. But some rules, like modus ponens, are more useful because they reflect the way that we reason informally. We'll list here some other inference rules that will come in handy when constructing proofs.

Modus Tollens (MT)	$\dfrac{A \to B, \neg B}{\therefore \neg A}.$	(6.8)
Conjunction (Conj)	$\dfrac{A, B}{\therefore A \wedge B}.$	(6.9)
Simplification (Simp)	$\dfrac{A \wedge B}{\therefore A},$	(6.10)
Addition (Add)	$\dfrac{A}{\therefore A \vee B}.$	(6.11)
Disjunctive Syllogism (DS)	$\dfrac{A \vee B, \neg A}{\therefore B}.$	(6.12)
Hypothetical Syllogism (HS)	$\dfrac{A \to B, B \to C}{\therefore A \to C}.$	(6.13)
Constructive Dilemma (CD)	$\dfrac{A \vee B, A \to C, B \to D}{\therefore C \vee D}.$	(6.14)
Destructive Dilemma (DD)	$\dfrac{\neg C \vee \neg D, A \to C, B \to D}{\therefore \neg A \vee \neg B}.$	(6.15)

Axioms

For any reasoning system to work, it needs some fundamental truths to start the process. An *axiom* is a wff that we wish to use as a basis from which to reason. So an axiom is usually a wff that we "know to be true" from our initial investigations (e.g., a proposition that has been shown to be a tautology by a truth table). When we apply logic to a particular subject, then an axiom might also be something that we "want to be true" to start out our discussion—for example, "Two points lie on one and only one line," for a geometry reasoning system.

We've introduced the three ingredients that make up any *formal reasoning system*:

a set of wffs, a set of axioms, a set of inference rules.

A formal reasoning system is often called a *formal theory.* How do we reason in such a system? Can we describe the reasoning process in some reasonable way? What is a proof? What is a theorem? Before we answer these questions, we should note that the formal theory that we are discussing in this section is often called a *natural deduction system* because we allow a wide variety of inference rules and when we know some wff is a tautology, then we can use it as an axiom. This allows us to introduce reasoning in a formal way but still keep the flavor of the way we reason informally. In Section 6.4 we'll discuss systems that have just one inference rule, modus ponens, and very few axioms, but for which reasoning can be quite difficult.

6.3.2 Formal Proof

Up to now our idea of proofs has been informal. We wrote sentences in English mixed with symbols and expressions from some domain of discourse and we made conclusions. Now we're going to focus on the structure of proof. So we better agree on definitions of proof and theorem.

Definition of Proof and Theorem

A *proof* is a finite sequence of wffs such that each wff in the sequence is either an axiom or can be inferred from previous wffs in the sequence. The last wff in a proof is called a *theorem.*

For example, suppose the following sequence of wffs is a proof.

$$W_1, \ldots, W_n.$$

Then we can say that W_n is a theorem because it is the last wff in the proof. We can also say that W_1 is an axiom because the first wff cannot be inferred from prior wffs in the sequence.

Conditional Proof

Most informal proofs do not satisfy this definition of proof. Instead, we usually start from one or more premises (or hypotheses) and then proceed to a conclusion by treating the premises just like axioms. For example, we assume that some statement A is true and then prove that some statement B follows from A. After such an argument we say "If A then B is true" or "From the premise A, we can conclude B." Let's make a formal definition for this kind of proof.

Proof from Premises

A *proof* (*or deduction*) *of B from a set of premises* is a finite sequence of wffs ending with B such that each wff in the sequence is either a premise, an axiom, or can be inferred from previous wffs in the sequence. A proof of B from an empty set of premises is a proof of B.

The nice thing about propositional calculus is that there is a result that allows us to prove conditionals like $A \rightarrow B$ by letting A be a premise and then constructing a proof of B. The result is called the *conditional proof rule* (CP). It is also known as the deduction theorem.

Conditional Proof Rule **(6.16)**

If there is a proof of B that uses A as a premise, then there is a proof of the conditional $A \to B$ that does not use A as a premise.

In a more general way, if there is a proof of B that uses premises $A_1, \ldots, A_n$, then there is a proof of $A_1 \wedge \cdots \wedge A_n \to B$ that does not use these premises.

We'll prove the CP rule later when we discuss specific axiom systems. But we can see that the general part (the second sentence) follows easily from the first sentence. For example, suppose we have a proof of B that uses two premises, A_1 and A_2. Since the proof uses the premise A_2, the CP rule tells us that there is a proof of $A_2 \to B$ that does not use A_2 as a premise. But it may still use A_1 as a premise. So we can use the CP rule to tell us that there is a proof of $A_1 \to (A_2 \to B)$ that does not use the two premises A_1 and A_2. Notice further (by an exercise in Section 6.2) that $A_1 \to (A_2 \to B) \equiv A_1 \wedge A_2 \to B$.

So the conditional proof rule is a useful tool for proving conditionals. For example, suppose we want to prove a conditional like $A_1 \wedge \cdots \wedge A_n \to B$. We can proceed as we do with informal proofs. Start the proof by listing the wffs $A_1, \ldots, A_n$ as premises. Then construct a proof whose last wff is B. Then we can use the CP rule to conclude that there is a proof of $A_1 \wedge \cdots \wedge A_n \to B$ that does not use any premises. So we conclude that $A_1 \wedge \cdots \wedge A_n \to B$ is a theorem.

Consistency

Suppose we are unlucky enough to have a formal theory with a wff W such that both W and $\neg W$ can be proved as theorems. A formal theory exhibiting this bad behavior is said to be *inconsistent.* We probably would agree that inconsistency is a bad situation. A formal theory that doesn't possess this bad behavior is said to be *consistent.* We certainly would like our formal theories to be consistent. For the propositional calculus we'll get consistency if we choose our axioms to be tautologies and we choose our inference rules to map tautologies to tautologies. Then every theorem will have to be a tautology.

Notation for Proofs

We'll write proofs in table format, where each line is numbered and contains a wff together with the reason it's there. For example, a proof sequence

$$W_1, \ldots, W_n$$

will be written in the following form:

Proof:	1.	W_1	Reason for W_1
	2.	W_2	Reason for W_2
	$\vdots$	$\vdots$	$\vdots$
	n.	W_n	Reason for W_n

The reason column for each line always contains a short indication of why the wff is on the line. If the line depends on previous lines because of an inference rule, then we'll always include the line numbers of those previous lines.

If a proof contains premises, then we'll indicate the fact that a line contains a premise by writing the letter

$$P.$$

After such a proof reaches the conclusion, the next line will contain QED along with the line numbers of the premises followed by CP. Here's an example to demonstrate the structure and notation that we'll be using.

example

6.8 A Proof from Premises

We'll give a proof of the following conditional wff.

$$(A \vee B) \wedge (A \vee C) \wedge \neg A \rightarrow B \wedge C.$$

The three premises are $A \vee B$, $A \vee C$, and $\neg A$. So we'll list them as premises to start the proof. Then we'll construct a proof of $B \wedge C$.

Proof:	1.	$A \vee B$	P
	2.	$A \vee C$	P
	3.	$\neg A$	P
	4.	B	1, 3, DS
	5.	C	2, 3, DS
	6.	$B \wedge C$	4, 5, Conj
		QED	1, 2, 3, 6, CP.

end example

Subproofs

Sometimes we need to prove a conditional as part of another proof. We'll call a proof that is part of another proof a *subproof* of the proof. We'll indicate a subproof that uses premises by indenting the wffs on its lines. We'll always write the conditional to be proved on the next line of the proof without indentation because it will be needed as part of the main proof. Let's do an example to show the idea.

6.9 A Subproof in a Proof

We'll give a proof of the following statement.

$$((A \vee B) \to (B \wedge C)) \to (B \to C) \vee D.$$

This wff is a conditional and the conclusion also contains a conditional. So the proof will contain a subproof of the conditional $B \to C$.

Proof:	1.	$(A \vee B) \to (B \wedge C)$	P	
	2.	$\quad B$	P	Start subproof of $B \to C$
	3.	$\quad A \vee B$	2, Add	
	4.	$\quad B \wedge C$	1, 3, MP	
	5.	$\quad C$	4, Simp	
	6.	$B \to C$	2, 5, CP	Finish subproof of $B \to C$
	7.	$(B \to C) \vee$ D	6, Add	
		QED	1, 7, CP.	

An Important Rule about Subproofs

If there is a proof from premises as a subproof within another proof, as indicated by the indented lines, then these indented lines may not be used to infer some line that occurs after the subproof is finished. The only exception to this rule is if an indented line does not depend, either directly or indirectly, on any premise of the subproof. In this case the indented line could actually be placed above the subproof, with the indentation removed.

6.10 A Sample Proof Structure

The following sequence of lines indicates a general proof structure for some conditional statement having the form $A \to M$. In the reason column for each line we have listed the possible lines that might be used to infer the given line.

1.	A			P
2.	B			1
3.		C		P
4.		D		1, 2, 3
5.			E	P
6.			F	1, 2, 3, 4, 5
7.			G	1, 2, 3, 4, 5, 6
8.		$E \rightarrow G$		5, 7, CP
9.		H		1, 2, 3, 4, 8
10.	$C \rightarrow H$			3, 9, CP
11.	I			1, 2, 10
12.		J		P
13.		K		1, 2, 10, 11, 12
14.		L		1, 2, 10, 11, 12, 13
15.	$J \rightarrow L$			12, 14, CP
16.	M			1, 2, 10, 11, 15
	QED			1, 16, CP.

end example

Simplifications in Proofs

We can make some simplifications in our proofs to reflect the way informal proofs are written. If W is a theorem, then we can use it to prove other theorems. We can put W on a line of a proof and treat it as an axiom. Or, better yet, we can leave W out of the proof sequence but still use it as a reason for some line of the proof.

example **6.11 Using Previous Results**

Let's prove the following statement.

$$\neg (A \wedge B) \wedge (B \vee C) \wedge (C \rightarrow D) \rightarrow (A \rightarrow D).$$

Proof:	1.	$\neg (A \wedge B)$		P
	2.	$B \vee C$		P
	3.	$C \rightarrow D$		P
	4.	$\neg A \vee \neg B$		1, $\neg (A \wedge B) \equiv \neg A \vee \neg B$
	5.		A	P

Proof:	6.	$\neg B$	4, 5, DS
	7.	C	2, 6, DS
	8.	D	3, 7, MP
	9.	$A \to D$	5, 8, CP
		QED	1, 2, 3, 9, CP.

Line 4 of the proof is OK because of the equivalence we listed in the reason column.

end example

Instead of writing down the specific theorem in the reason column, we'll usually write the symbol

$$T$$

to indicate that a theorem is being used.

Some proofs are straightforward, while others can be brain busters. Remember, when you construct a proof, it may take several false starts before you come up with a correct proof sequence. Let's do some more examples.

example 6.12 Using Previous Results

We will give a proof of the following wff:

$$A \wedge ((A \to B) \vee (C \wedge D)) \to (\neg B \to C).$$

Proof:	1.	A	P
	2.	$(A \to B) \vee (C \wedge D)$	P
	3.	$\neg B$	P
	4.	$A \wedge \neg B$	1, 3, Conj
	5.	$\neg \neg A \wedge \neg B$	4, T
	6.	$\neg (\neg A \vee B)$	5, T
	7.	$\neg (A \to B)$	6, T
	8.	$C \wedge D$	2, 7, DS
	9.	C	8, Simp
	10.	$\neg B \to C$	3, 9, CP
		QED	1, 2, 10, CP.

end example

example 6.13 Formalizing an Argument

Let's consider the argument given by the following sentences.

The team wins or I am sad. If the team wins, then I go to a movie. If I am sad, then my dog barks. My dog is quiet. Therefore, I go to a movie.

To formalize this argument, we'll make the following substitutions.

W:	The team wins.
S:	I am sad.
M:	I go to a movie.
B:	My dog barks.

Now we can symbolize the argument with the following wff:

$$(W \vee S) \wedge (W \to M) \wedge (S \to B) \wedge \neg B \to M.$$

We'll show this wff is a theorem with the following proof.

Proof:	1.	$W \vee S$	P
	2.	$W \to M$	P
	3.	$S \to B$	P
	4.	$\neg B$	P
	5.	$\neg S$	3, 4, MT
	6.	W	1, 5, DS
	7.	M	2, 6, MP
		QED	1, 2, 3, 4, 7, CP.

end example

Indirect Proof

Suppose we want to prove a statement, but we just can't seem to find a way to get going. We might try *proof by contradiction* or *reductio ad absurdum.* In other words, assume that the statement to be proven is false and argue from there until a contradiction of some kind is reached. The idea is based on the following equivalence for any wff W:

$$W \equiv \neg W \to \text{false}.$$

So to prove W it suffices to prove the conditional $\neg W \to$ false. Here's the formal description of the indirect proof rule.

Indirect Proof Rule (IP) (6.17)

To prove the wff W, construct a proof of $\neg W \to$ false. In particular, to prove $A \to B$, construct a proof of $A \wedge \neg B \to$ false.

Proof: The first sentence follows from the preceding discussion. The second sentence follows from the equivalence

$$\neg (A \to B) \equiv A \wedge \neg B.$$

For then we have $A \to B \equiv \neg (A \to B) \to \text{false} \equiv A \wedge \neg B \to \text{false}$. QED.

Proof by contradiction is nice because we always get an extra premise from which to work. For example, to prove the conditional $A \to B$, we take A as a premise and we take $\neg B$ as an additional premise. We also have more freedom because all we need to do is find any kind of contradiction. You might try it whenever there doesn't seem to be enough information from the given premises or when you run out of ideas. You might also try it as your first method of proof.

Notation for Indirect Proofs

When we prove a wff W with the IP rule, we'll write $\neg W$ as a premise on a line along with "P for IP" to indicate the reason. If W is a conditional $A \to B$, then we'll write A as a premise on one line, and we'll write B as a "P for IP" on another line. After the proof reaches a contradiction, the next line will contain QED along with the line numbers of the premises followed by IP.

example 6.14 An Indirect Proof

We'll give an indirect proof of the following statement from Example 6.13:

$$(W \vee S) \wedge (W \to M) \wedge (S \to B) \wedge \neg B \to M.$$

Since the statement to be proved is a conditional, we'll start the proof by writing down the four premises followed by $\neg M$ as the premise for IP.

Proof:	1.	$W \vee S$	P
	2.	$W \to M$	P
	3.	$S \to B$	P
	4.	$\neg B$	P
	5.	$\neg M$	P for IP
	6.	$\neg W$	2, 5, MT
	7.	$\neg S$	3, 4, MT
	8.	$\neg W \wedge \neg S$	6, 7, Conj
	9.	$\neg (W \vee S)$	8, T
	10.	$(W \vee S) \wedge \neg (W \vee S)$	1, 9, Conj
	11.	false	10, T
		QED	1–4, 5, 11, IP.

end example

Compare this proof to the earlier direct proof of the same statement. It's a bit longer, and it uses different rules. Sometimes a longer proof can be easier to create, using simpler steps. Just remember, there's more than one way to proceed when trying a proof.

6.3.3 Proof Notes

When proving something, we should always try to *tell the proof, the whole proof, and nothing but the proof.* Here are a few concrete suggestions that should make life easier for beginning provers.

Don't Use Unnecessary Premises

Sometimes beginners like to put extra premises in proofs to help get to a conclusion. But then they forget to give credit to these extra premises. For example, suppose we want to prove a conditional of the form $A \rightarrow C$. We start the proof by writing A as a premise. Suppose that along the way we decide to introduce another premise, say B, and then use A and B to infer C, either directly or indirectly. The result is not a proof of $A \rightarrow C$. Instead, we have given a proof of the statement $A \wedge B \rightarrow C$.

Remember: Be sure to use a premise only when it's the hypothesis of a conditional that you want to prove. Another way to say this is: If you use a premise to prove something, then the premise becomes part of the antecedent of the thing you proved. Still another way to say this is:

Premises

The conjunction of all the premises that you used to prove something is precisely the antecedent of the conditional that you proved.

Don't Apply Inference Rules to Subexpressions

Beginners sometimes use an inference rule incorrectly by applying it to a subexpression of a larger wff. This violates the definition of a proof, which states that a wff in a proof either is an axiom or is inferred by previous wffs in the proof. In other words, an inference rule can be applied only to entire wffs that appear on previous lines of the proof. So here it is for the record:

Subexpressions

Don't apply inference rules to subexpressions of wffs.

Let's do an example to see what we are talking about. Suppose we have the following wff:

$$A \wedge ((A \rightarrow B) \vee C) \rightarrow B.$$

This wff is not a tautology. For example, let A = true, B = false, and C = true. The following sequence attempts to show that the wff is a theorem:

1.	A	P	
2.	$(A \rightarrow B) \vee C$	P	
3.	B	1, 2, MP	Incorrect use of MP
	QED	1, 2, 3, CP.	

The reason that the proof is wrong is that MP is applied to the two wffs A on line 1 and $A \rightarrow B$ on line 2. But $A \rightarrow B$ does not occur on a line by itself. Rather, it's a subexpression of $(A \rightarrow B) \vee C$. Therefore, MP cannot be used.

Failure to Find a Proof

Sometimes we can obtain valuable information about a wff by failing to find a proof. After a few unsuccessful attempts, it may dawn on us that the thing is not a theorem. For example, no proof exists for the following conditional wff:

$$(A \rightarrow C) \rightarrow (A \vee B \rightarrow C).$$

To see that this wff is not a tautology, let A = false, C = false, and B = true.

But remember that we cannot conclude that a wff is not a tautology just because we can't find a proof.

Exercises

Proof Structures

1. Let W denote the wff $(A \rightarrow B) \rightarrow B$. It's easy to see that W is not a tautology. Just let B = false and A = false. Now, suppose someone claims that the following sequence of statements is a "proof" of W:

1.	$A \rightarrow B$	P
2.	A	P
3.	B	1, 2, MP
	QED	1, 3, CP.

   a. What is wrong with the above "proof" of W?

   b. Write down the statement that the "proof" proves.

2. Let W denote the wff $(A \rightarrow (B \wedge C)) \rightarrow (A \rightarrow B) \wedge C$. It's easy to see that W is not a tautology. Suppose someone claims that the following sequence of statements is a "proof" of W:

1.	$A \to (B \wedge C)$	P
2.	A	P
3.	$B \wedge C$	1, 2, MP
4.	B	3, Simp
5.	$A \to B$	2, 4, CP
6.	C	3, Simp
7.	$(A \to B) \wedge C$	5, 6, Conj
	QED	1, 7, CP.

What is wrong with this "proof" of W?

3. Find the number of premises required for a proof of each of the following wffs. Assume that the letters stand for other wffs.

   a. $A \to (B \to (C \to D))$.
   b. $((A \to B) \to C) \to D$.

4. Give a formalized version of the following proof.

   If I am dancing, then I am happy. There is a mouse in the house or I am happy. I am sad. Therefore, there is a mouse in the house and I am not dancing.

Formal Proofs

5. Give formal proofs for each of the following tautologies by using the CP rule.

   a. $A \to (B \to (A \wedge B))$.
   b. $A \to (\neg B \to (A \wedge \neg B))$.
   c. $(A \vee B \to C) \wedge A \to C$.
   d. $(B \to C) \to (A \wedge B \to A \wedge C)$.
   e. $(A \vee B \to C \wedge D) \to (B \to D)$.
   f. $(A \vee B \to C) \wedge (C \to D \wedge E) \to (A \to D)$.
   g. $\neg (A \wedge B) \wedge (B \vee C) \wedge (C \to D) \to (A \to D)$.
   h. $(A \to (B \to C)) \to (B \to (A \to C))$.
   i. $(A \to C) \to (A \wedge B \to C)$.
   j. $(A \to C) \to (A \to B \vee C)$.

6. Give formal proofs for each of the following tautologies by using the IP rule.

   a. $A \to (B \to A)$.
   b. $(A \to B) \wedge (A \vee B) \to B$.
   c. $\neg B \to (B \to C)$.
   d. $(A \to B) \to (C \vee A \to C \vee B)$.

e. $(A \to B) \to ((A \to \neg B) \to \neg A)$.
f. $(A \to B) \to ((B \to C) \to (A \vee B \to C))$.
g. $(A \to B) \wedge (B \to C) \to (A \to C)$. *Note:* This is the HS inference rule.
h. $(C \to A) \wedge (\neg C \to B) \to (A \vee B)$.

7. Give formal proofs for each of the following tautologies by using the IP rule somewhere in each proof.

   a. $A \to (B \to (A \wedge B))$.
   b. $A \to (\neg B \to (A \wedge \neg B))$.
   c. $(A \vee B \to C) \wedge A \to C$.
   d. $(B \to C) \to (A \wedge B \to A \wedge C)$.
   e. $(A \vee B \to C \wedge D) \to (B \to D)$.
   f. $(A \vee B \to C) \wedge (C \to D \wedge E) \to (A \to D)$.
   g. $\neg (A \wedge B) \wedge (B \vee C) \wedge (C \to D) \to (A \to D)$.
   h. $(A \to B) \to ((B \to C) \to (A \vee B \to C))$.
   i. $(A \to (B \to C)) \to (B \to (A \to C))$.
   j. $(A \to C) \to (A \wedge B \to C)$.
   k. $(A \to C) \to (A \to B \vee C)$.

8. Give a formal proof of the equivalence $A \wedge B \to C \equiv A \to (B \to C)$. In other words, prove both of the following statements. Use either CP or IP.

   a. $(A \wedge B \to C) \to (A \to (B \to C))$.
   b. $(A \to (B \to C)) \to (A \wedge B \to C)$.

Challenges

9. Give a formal proof using the other inference rules for each of the dilemma inference rules.

   a. Constructive dilemma (CD).
   b. Destructive dilemma (DD).

10. Two students came up with the following different wffs to reflect the meaning of the statement "If A then B else C."

$$(A \to B) \wedge (\neg A \to C)$$
$$(A \wedge B) \vee (\neg A \wedge C)$$

Prove that the two wffs are equivalent by finding formal proofs for the following two statements.

   a. $((A \to B) \wedge (\neg A \to C)) \to ((A \wedge B) \vee (\neg A \wedge C))$.
   b. $((A \wedge B) \vee (\neg A \wedge C)) \to ((A \to B) \wedge (\neg A \to C))$.

6.4 Formal Axiom Systems

Although truth tables are sufficient to decide the truth of a propositional wff, most of us do not reason by truth tables. Instead, we use our personal reasoning systems. In the proofs presented up to now we allowed the use of any of the inference rules (6.7) to (6.13). Similarly, we allowed the use of any known tautology as an axiom. This natural deduction system is a loose kind of formal system for the propositional calculus.

Can we find a proof system for the propositional calculus that contains a specific set of axioms and inference rules? If we do find such a system, how do we know that it does the job? In fact, what is the job that we want done? Basically, we want two things. We want our proofs to yield theorems that are tautologies, and we want any tautology to be provable as a theorem. These properties are converses of each other, and they have the following names.

Soundness: All proofs yield theorems that are tautologies.

Completeness: All tautologies are provable as theorems.

So if we want to reason in a theory for which our theorems are actually true, then we must ensure that we use only inference rules that map true statements to true statements. Of course, we must also ensure that the axioms that we choose are true.

6.4.1 An Example Axiom System

Is there a simple formal system for which we can show that the propositional calculus is both sound and complete? Yes, there is. In fact, there are many of them. Each one specifies a small fixed set of axioms and inference rules. The pioneer in this area was the mathematician Gottlob Frege (1848–1925). He formulated the first such axiom system [1879]. We'll discuss it further in the exercises. Later, in 1930, J. Lukasiewicz showed that Frege's system, which has six axioms, could be replaced by the following three axioms, where A, B, and C can represent any wff generated by propositional variables and the two connectives $\neg$ and $\rightarrow$.

Frege-Lukasiewicz Axioms **(6.18)**

1. $A \rightarrow (B \rightarrow A)$.
2. $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$.
3. $(\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)$.

The only inference rule used is modus ponens. Although the axioms may appear a bit strange, they can all be verified by truth tables. Also note that conjunction and disjunction are missing. But this is no problem, since we know that they can be written in terms of implication and negation.

We'll use this system to prove the CP rule (i.e., the deduction theorem). But first we must prove a result that we'll need, namely that $A \to A$ is a theorem provable from the axioms. Notice that the proof uses only the given axioms and modus ponens.

Lemma 1. $A \to A$.

Proof:			
	1.	$(A \to ((B \to A) \to A)) \to ((A \to (B \to A)) \to (A \to A))$	Axiom 2
	2.	$A \to ((B \to A) \to A)$	Axiom 1
	3.	$(A \to (B \to A)) \to (A \to A)$	1, 2, MP
	4.	$A \to (B \to A)$	Axiom 1
	5.	$A \to A$	3, 4, MP.
		QED	

Now we're in position to prove the CP rule, which is also called the deduction theorem. The proof, which is due to Herbrand (1930), uses only the given axioms, modus ponens, and Lemma 1.

Deduction Theorem (Conditional Proof Rule, CP)

If A is a premise in a proof of B, then there is a proof of $A \to B$ that does not use A as a premise.

Proof: Assume that A is a premise in a proof of B. We must show that there is a proof of $A \to B$ that does not use A as a premise. Suppose that $B_1, \ldots, B_n$ is a proof of B that uses the premise A. We'll show by induction that for each k in the interval $1 \leq k \leq n$, there is a proof of $A \to B_k$ that does not use A as a premise. Since $B = B_n$, the result will be proved. If $k = 1$, then either $B_1 = A$ or B_1 is an axiom or a premise other than A. If $B_1 = A$, then $A \to B_1 = A \to A$, which by Lemma 1 has a proof that does not use A as a premise. If B_1 is an axiom or a premise other than A, then the following proof of $A \to B_1$ does not use A as a premise.

1.	B_1	An axiom or premise other than A
2.	$B_1 \to (A \to B_1)$	Axiom 1
3.	$A \to B_1$	1, 2, MP.

Now assume that for each $i < k$ there is a proof of of $A \to B_i$ that does not use A as a premise. We must show that there is a proof of $A \to B_k$ that does not use A as a premise. If $B_k = A$ or B_k is an axiom or a premise other than A, then we can use the same argument for the case when $k = 1$ to conclude that there is a proof of $A \to B_k$ that does not use A as a premise. If B_k is not an axiom or a premise, then it is inferred by MP from two wffs in the proof of the form B_i and $B_j = B_i \to B_k$, where $i < k$ and $j < k$. Since by $i < k$ and $j < k$, the induction hypothesis tells us that there are proofs of $A \to B_i$ and $A \to (B_i \to B_k)$, neither of which contains A as a premise. Now consider the following proof, where lines 1 and 2 represent the proofs of $A \to B_i$ and $A \to (B_i \to B_k)$.

1.	Proof of $A \to B_i$	Induction
2.	Proof of $A \to (B_i \to B_k)$	Induction
3.	$(A \to (B_i \to B_k)) \to ((A \to B_i) \to (A \to B_k))$	Axiom 2
4.	$(A \to B_i) \to (A \to B_k)$	2, 3, MP
5.	$A \to B_k$	1, 4, MP.

So there is a proof of $A \to B_k$ that does not contain A as a premise. Let $k = n$ to obtain the desired result because $B_n = B$. QED.

Once we have the CP rule, proofs become much easier since we can have premises in our proofs. But still, everything that we do must be based only on the axioms, MP, CP, and any results we prove along the way. The system is sound because the axioms are tautologies and MP maps tautologies to a tautology. In other words, every proof yields a theorem that is a tautology.

The remarkable thing is that this little axiom system is complete in the sense that there is a proof within the system for every tautology of the propositional calculus.

We'll give a few more examples to get the flavor of reasoning from a very small set of axioms.

example 6.15 Hypothetical Syllogism

We'll use CP to prove the hypothetical syllogism rule of inference.

Hypothetical Syllogism (HS) **(6.19)**

From the premises $A \to B$ and $B \to C$, there is a proof of $A \to C$.

Proof:	1.	$A \to B$	P
	2.	$B \to C$	P
	3.	A	P
	4.	B	1, 3, MP
	5.	C	2, 4, MP
	6.	$A \to C$	3, 5, CP.
		QED	

We can apply the CP rule to HS to say that from the premise $A \to B$ there is a proof of $(B \to C) \to (A \to C)$. One more application of the CP rule tells us that the following wff has a proof with no premises:

$$(A \to B) \to ((B \to C) \to (A \to C)).$$

This is just another way to represent the HS rule as a wff.

end example

Six Sample Proofs

Now that we have CP and HS rules, it should be easier to prove statements within the axiom system. Let's prove the following six statements.

Six Tautologies **(6.20)**

a. $\neg A \to (A \to B)$.

b. $\neg \neg A \to A$.

c. $A \to \neg \neg A$.

d. $(A \to B) \to (\neg B \to \neg A)$.

e. $A \to (\neg B \to \neg (A \to B))$.

f. $(A \to B) \to ((\neg A \to B) \to B)$.

In the next six examples we'll prove these six statements using only the axioms, MP, HS, and previously proven results.

example **6.16 (a) $\neg A \to (A \to B)$**

Proof:	1.	$\neg A$	P
	2.	$\neg A \to (\neg B \to \neg A)$	Axiom 1
	3.	$\neg B \to \neg A$	1, 2, MP
	4.	$(\neg B \to \neg A) \to (A \to B)$	Axiom 3
	5.	$A \to B$	3, 4, MP
		QED	1, 5, CP.

end example

example **6.17 (b) $\neg\neg A \rightarrow A$**

Proof:	1.	$\neg\neg A$	P
	2.	$\neg\neg A \rightarrow (\neg A \rightarrow \neg\neg\neg A)$	Part (a)
	3.	$\neg A \rightarrow \neg\neg\neg A$	1, 2, MP
	4.	$(\neg A \rightarrow \neg\neg\neg A) \rightarrow (\neg\neg A \rightarrow A)$	Axiom 3
	5.	$\neg\neg A \rightarrow A$	3, 4, MP
	6.	A	1, 5, MP
		QED	1, 6, CP.

end example

example **6.18 (c) $A \rightarrow \neg\neg A$**

Proof:	1.	$\neg\neg\neg A \rightarrow \neg A$	Part (b)
	2.	$(\neg\neg\neg A \rightarrow \neg A) \rightarrow (A \rightarrow \neg\neg A)$	Axiom 3
	3.	$A \rightarrow \neg\neg A$	1, 2, MP
		QED	1, 3, CP.

end example

example **6.19 (d) $(A \rightarrow B) \rightarrow (\neg B \rightarrow \neg A)$**

Proof:	1.	$A \rightarrow B$	P
	2.	$\neg\neg A \rightarrow A$	Part (b)
	3.	$\neg\neg A \rightarrow B$	1, 2, HS
	4.	$B \rightarrow \neg\neg B$	Part (c)
	5.	$\neg\neg A \rightarrow \neg\neg B$	3, 4, HS
	6.	$(\neg\neg A \rightarrow \neg\neg B) \rightarrow (\neg B \rightarrow \neg A)$	Axiom 3
	7.	$\neg B \rightarrow \neg A$	5, 6, MP
		QED	1, 7, CP.

end example

example **6.20 (e)** $\boldsymbol{A \to (\neg B \to \neg (A \to B))}$

Proof:			
	1.	A	P
	2.	$\quad A \to B$	P
	3.	$\quad B$	1, 2, MP
	4.	$(A \to B) \to B$	2, 3, CP
	5.	$((A \to B) \to B) \to (\neg B \to \neg (A \to B))$	Part (d)
	6.	$\neg B \to \neg (A \to B)$	4, 5, MP
		QED	1, 6, CP.

end example

example **6.21 (f)** $\boldsymbol{(A \to B) \to ((\neg A \to B) \to B)}$

Proof:			
	1.	$A \to B$	P
	2.	$(A \to B) \to (\neg B \to \neg A)$	Part (d)
	3.	$\neg B \to \neg A$	1, 2, MP
	4.	$\quad \neg A \to B$	P
	5.	$\quad \neg B \to B$	3, 4, HS
	6.	$\quad \neg B \to (B \to \neg (A \to B))$	Part (a)
	7.	$\quad (\neg B \to (B \to \neg (A \to B)))$	
		$\qquad \to ((\neg B \to B) \to (\neg B \to \neg (A \to B)))$	Axiom 2
	8.	$\quad (\neg B \to B) \to (\neg B \to \neg (A \to B))$	6, 7, MP
	9.	$\quad \neg B \to \neg (A \to B)$	5, 8, MP
	10.	$\quad (\neg B \to \neg (A \to B)) \to ((A \to B) \to B))$	Axiom 3
	11.	$\quad (A \to B) \to B)$	9, 10, MP
	12.	$\quad B$	1, 11, MP
	13.	$(\neg A \to B) \to B$	4, 12, CP
		QED	1, 13, CP.

end example

Completeness of the Axiom System

As we mentioned earlier, it is a remarkable result that this axiom system is complete. The interesting thing is that we now have enough tools to find a proof of completeness.

Lemma 2

Let W be a wff with propositional variables $P_1,..., P_m$. For any truth assignment to the variables, let $Q_1,..., Q_m$ be defined by letting each Q_k be either P_k or $\neg P_k$ depending on whether P_k is assigned true or false, respectively. Then from the premises $Q_1,..., Q_m$ there is a proof of either W or $\neg W$ depending on whether the assignment makes W true or false, respectively.

Proof: The proof is by induction on the number n of connectives that occcur in W. If $n = 0$, then W is just a propositional variable P. If P is assigned true, then we must find a proof of P using P as its own premise.

1.	P	Premise
2.	$P \to P$	Lemma 1.
3.	P	1, 2, MP
	QED	1, 3, CP.

If P is assigned false, then we must find a proof of $\neg P$ from premise $\neg P$.

1.	$\neg P$	Premise
2.	$\neg P \to \neg P$	Lemma 1.
3.	$\neg P$	1, 2, MP
	QED	1, 3, CP.

So assume W is a wff with n connectives where $n > 0$ and assume that the lemma is true for all wffs with less than n connectives. Now W has one of two forms, $W = \neg A$ or $W = A \to B$. It follows that A and B each have less than n connectives, so by induction there are proofs from the premises $Q_1,\ldots, Q_m$ of either A or $\neg A$ and of either B or $\neg B$ depending on whether they are made true or false by the truth assignment. We'll argue for each form of W.

Let $W = \neg A$. If W is true, then A is false, so there is a proof from the premises of $\neg A = W$. If W is false, then A is true, so there is a proof from the premises of A. Now (6.20c) gives us $A \to \neg \neg A$. So by MP there is a proof from the premises of $\neg \neg A = \neg W$.

Let $W = A \to B$. Assume W is true. Then either A is false or B is true. If A is false, then there is a proof from the premises of $\neg A$. Now (6.20a) gives us $\neg A \to (A \to B)$. So by MP there is a proof from the premises of $(A \to B) = W$. If B is true, then there is a proof from the premises of B. Now Axiom 1 gives us $B \to (A \to B)$. So by MP there is a proof from the premises of $(A \to B) = W$.

Assume W is false. Then A is true and B is false. So there is a proof from the premises of A and there is a proof from the premises of $\neg B$. Now (6.20e) gives us $A \to (\neg B \to \neg (A \to B))$. So by two applications of MP there is a proof from the premises of $(A \to B) = W$. QED.

Theorem (Completeness)

Any tautology can be proven as a theorem in the axiom system.

Proof: Let W be a tautology with propositional variables $P_1, \ldots, P_m$. Since W is always true, it follows from Lemma 2 that for any truth assignment to the propositional variables $P_1, \ldots, P_m$ that occur in W, there is a proof of W from the premises $Q_1, \ldots, Q_m$, where each Q_k is either P_k or $\neg P_k$ according to whether the truth assignment to P_k is true or false, respectively. Now if P_m is assigned true, then $Q_m = P_m$ and if P_m is assigned false, then $Q_m = \neg P_m$. So there are two proofs of W, one with premises $Q_1, \ldots, Q_{m-1}, P_m$ and one with premises $Q_1, \ldots, Q_{m-1}, \neg P_m$. Now apply the CP rule to both proofs to obtain the following two proofs.

A proof from premises $Q_1, \ldots, Q_{m-1}$ of $(P_m \rightarrow W)$.

A proof from premises $Q_1, \ldots, Q_{m-1}$ of $(\neg P_m \rightarrow W)$.

We combine the two proofs into one proof. Now (6.20f) gives us the following statement, which we add to the proof:

$$(P_m \rightarrow W) \rightarrow ((\neg P_m \rightarrow W) \rightarrow W).$$

Now with two applications of MP, we obtain W. So there is a proof of W from premises $Q_1, \ldots, Q_{m-1}$. Now use the same procedure for $m - 1$ more steps to obtain a proof of W with no premises. QED.

6.4.2 Other Axiom Systems

There are many other small axiom systems for the propositional calculus that are sound and complete. For example, Frege's original axiom system consists of the following six axioms together with the modus ponens inference rule, where A, B, and C can represent any wff generated by propositional variables and the two connectives $\neg$ and $\rightarrow$.

Frege's Axioms **(6.21)**

1. $A \rightarrow (B \rightarrow A)$.
2. $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$.
3. $(A \rightarrow (B \rightarrow C)) \rightarrow (B \rightarrow (A \rightarrow C))$.
4. $(A \rightarrow B) \rightarrow (\neg B \rightarrow \neg A)$.
5. $\neg \neg A \rightarrow A$.
6. $A \rightarrow \neg \neg A$.

If we examine the proof of the CP rule, we see that it depends only on Axioms 1 and 2 of (6.18), which are the same as Frege's first two axioms. Also, HS is proved with CP. So we can continue reasoning from these axioms with CP and HS in our tool kit. We'll discuss this system in the exercises.

Another small axiom system, due to Hilbert and Ackermann [1938], consists of the following four axioms together with the modus ponens inference rule, where $A \rightarrow B$ is used as an abreviation for $\neg A \vee B$, and where A, B, and C can represent any wff generated by propositional variables and the two connectives $\neg$ and $\vee$.

Hilbert-Ackermann Axioms **(6.22)**

1. $A \vee A \rightarrow A$.
2. $A \rightarrow A \vee B$.
3. $A \vee B \rightarrow B \vee A$.
4. $(A \rightarrow B) \rightarrow (C \vee A \rightarrow C \vee B)$.

We'll discuss this system in the exercises too.

There are important reasons for studying small formal systems like the axiom systems we've been discussing. Small systems are easier to test and easier to compare with other systems because there are only a few basic operations to worry about. For example, if we build a program to do automatic reasoning, it may be easier to implement a small set of axioms and inference rules. This also applies to computers with small instruction sets and to programming languages with a small number of basic operations.

Exercises

Axiom Systems

1. In Frege's axiom system (6.21), prove that Axiom 3 follows from Axioms 1 and 2. That is, prove the following statement from Axioms 1 and 2:

$$(A \rightarrow (B \rightarrow C)) \rightarrow (B \rightarrow (A \rightarrow C)).$$

2. In Frege's axiom system, prove that $(\neg A \rightarrow \neg B) \rightarrow (B \rightarrow A)$.

3. In the Hilbert-Ackermann axiom system (6.22), prove each of the following statements. Do not use the CP rule. You can use any previous theorems in the list to prove subsequent statements.

a. $(A \to B) \to ((C \to A) \to (C \to B))$.
b. (HS) If $A \to B$ and $B \to C$ are theorems, then $A \to C$ is a theorem.
c. $A \to A$ (i.e., $\neg A \vee A$).
d. $A \vee \neg A$.
e. $A \to \neg\neg A$.
f. $\neg\neg A \to A$.
g. $\neg A \to (A \to B)$.
h. $(A \to B) \to (\neg B \to \neg A)$.
i. $(\neg B \to \neg A) \to (A \to B)$.

Logic Puzzles

4. The county jail is full. The sheriff, Anne Oakley, brings in a newly caught criminal and decides to make some space for the criminal by letting one of the current inmates go free. She picks prisoners A, B, and C to choose from. She puts blindfolds on A and B because C is already blind. Next she selects three hats from five hats hanging on the hat rack, two of which are red and three of which are white, and places the three hats on the prisoner's heads. She hides the remaining two hats. Then she takes the blindfolds off A and B and tells them what she has done, including the fact that there were three white hats and two red hats to choose from. Sheriff Oakley then says, "If you can tell me the color of the hat you are wearing, without looking at your own hat, then you can go free." The following things happen:

   1. A says that he can't tell the color of his hat. So the sheriff has him returned to his cell.

   2. Then B says that he can't tell the color of his hat. So he is also returned to his cell.

   3. Then C, the blind prisoner, says that he knows the color of his hat. He tells the sheriff, and she sets him free.

   What color was C's hat, and how did C do his reasoning?

5. Four men and four women were nominated for two positions on the school board. One man and one woman were elected to the positions. Suppose the men are named A, B, C, and D and the women are named E, F, G, and H. Further, suppose that the following four statements are true:

   1. If neither A nor E won a position, then G won a position.

   2. If neither A nor F won a position, then B won a position.

   3. If neither B nor G won a position, then C won a position.

   4. If neither C nor F won a position, then E won a position.

   Who were the two people elected to the school board?

6.5 Chapter Summary

Propositional calculus is the basic building block of formal logic. Each wff represents a statement that can be checked by truth tables to determine whether it is a tautology, a contradiction, or a contingency. There are basic equivalences (6.1) that allow us to simplify and transform wffs into other wffs. We can use these equivalences with Quine's method to determine whether a wff is a tautology, a contradiction, or a contingency. We can also use the equivalences to transform any wff into a DNF or a CNF. Any truth function has one of these forms.

Propositional calculus also provides us with formal techniques for proving properties of wffs without using truth tables. A formal reasoning system has wffs, axioms, and inference rules. Some useful inference rules are modus ponens, modus tollens, conjunction, simplification, addition, disjunctive syllogism, hypothetical syllogism, constructive dilemma, and destructive dilemma. When constructing proofs, remember: *Don't use unnecessary premises*, and *don't apply inference rules to subexpressions.*

We want formal reasoning systems to be sound—proofs yield theorems that are tautologies—and complete—all tautologies can be proven as theorems. The system presented in this chapter is sound and complete as long as we always use tautologies as axioms. There are even small axiomatic systems for the propositional calculus that are sound and complete. We presented one that has three axioms and one inference rule.

Notes

The logical symbols that we've used in this chapter are not universal. So you should be flexible in your reading of the literature. From a historical point of view, Whitehead and Russell [1910] introduced the symbols $\supset$, $\vee$, $\cdot$, $\sim$, and $\equiv$ to stand for implication, disjunction, conjunction, negation, and equivalence, respectively. A prefix notation for the logical operations was introduced by Lukasiewicz [1929], where the letters C, A, K, N, and E stand for implication, disjunction, conjunction, negation, and equivalence, respectively. So in terms of our notation we have $Cpq = p \to q$, $Apq = p \vee q$, $Kpq = p \wedge q$, $Nq = \neg\, q$, and $Epq = p \equiv q$. This notation is called Polish notation, and its advantage is that each expression has a unique meaning without using parentheses and precedence. For example, $(p \to q) \to r$ and $p \to (q \to r)$ are represented by the expressions $CCpqr$ and $CpCqr$, respectively. The disadvantage of the notation is that it's harder to read. For example, $CCpqKsNr = (p \to q) \to (s \wedge \neg\, r)$.

The fact that a wff W is a theorem is often denoted by placing a turnstile in front of it as follows:

$$\vdash W.$$

So this means that there is a proof $W_1, \ldots, W_n$ such that $W_n = W$. Turnstiles are also used in discussing proofs that have premises. For example, the notation

$$A_1, A_2, \ldots, A_n \vdash B$$

means that there is a proof of B from the premises $A_1, A_2, \ldots, A_n$.

We should again emphasize that the logic that we are studying in this book deals with statements that are either true or false. This is sometimes called the *Law of the Excluded Middle*: Every statement is either true or false. If our logic does not assume the law of the excluded middle, then we can no longer use indirect proof because we can't conclude that a statement is false from the assumption that it is not true. A logic called *intuitionist logic* omits this law and thus forces all proofs to be direct. Intuitionists like to construct things in a direct manner.

Logics that assume the law of the excluded middle are called *two-valued logics*. Some logics take a more general approach and consider statements that may not be true or false. For example, a *three-valued logic* assigns one of three values to each statement: 0, .5, or 1, where 0 stands for false, .5 stands for unknown, and 1 stands for true. We can build truth tables for this logic by defining $\neg A = 1 - A$, $A \vee B = \max(A, B)$, and $A \wedge B = \min(A, B)$. We still use the equivalence $A \to B \equiv \neg A \vee B$. So we can discuss three-valued logic.

In a similar manner we can discuss *n-valued logic* for any natural number $n \geq 2$, where each statement takes on one of n specific values in the range 0 to 1. Some n-valued logics assign names to the values such as "necessarily true," "probably true," "probably false," and "necessarily false." For example, there is a logic called *modal logic* that uses two extra unary operators, one to indicate that a statement is necessarily true and one to indicate that a statement is possibly true. So modal logic can represent a sentence like "If P is necessarily true, then P is true."

Some logics assign values over an infinite set. For example, the term *fuzzy logic* is used to describe a logic in which each statement is assigned some value in the closed unit interval $[0, 1]$.

All these logics have applications in computer science but they are beyond our scope and purpose. However, it's nice to know that they all depend on a good knowledge of two-valued logic. In this chapter we've covered the fundamental parts of two-valued logic—the properties and reasoning rules of the propositional calculus. We'll see that these ideas occur in all the logics and applications that we cover in the next three chapters.

chapter 7

Predicate Logic

Error of opinion may be tolerated
where reason is left free to combat it.
—Thomas Jefferson (1743–1826)

We need a new logic if we want to describe arguments that deal with all cases or with some case out of many cases. In this chapter we'll introduce the notions and notations of first-order predicate calculus. This logic will allow us to analyze and symbolize a wider variety of statements and arguments than can be done with propositional logic.

chapter guide

Section 7.1 introduces the basic syntax and semantics of the first-order predicate calculus. We'll discuss the properties of validity and satisfiability, and we'll discuss the problem of deciding whether a formula is valid.

Section 7.2 introduces the fundamental equivalences of first-order predicate calculus. We'll discuss the prenex normal forms, and we'll look at the problem of formalizing English sentences.

Section 7.3 introduces the standard inference rules that allow us to do formal reasoning in first-order predicate calculus in much the same way that we do informal reasoning.

7.1 First-Order Predicate Calculus

The propositional calculus provides adequate tools for reasoning about propositional wffs, which are combinations of propositions. But a proposition is a sentence taken as a whole. With this restrictive definition, propositional calculus doesn't provide the tools to do everyday reasoning. For example, in the following argument it is impossible to find a formal way to test the correctness of the inference without further analysis of each sentence:

> All computer science majors own a personal computer.
> Socrates does not own a personal computer.
> Therefore, Socrates is not a computer science major.

To discuss such an argument, we need to break up the sentences into parts. The words in the set {All, own, not} are important to understand the argument. Somehow we need to symbolize a sentence so that the information needed for reasoning is characterized in some way. Therefore, we will study the inner structure of sentences.

7.1.1 Predicates and Quantifiers

The statement "x owns a personal computer" is not a proposition because its truth value depends on x. If we give x a value, like $x =$ Socrates, then the statement becomes a proposition because it has the value true or false. From a grammar point of view, the property "owns a personal computer" is a predicate, where a predicate is the part of a sentence that gives a property of the subject. A predicate usually contains a verb, like "owns" in our example. The word predicate comes from the Latin word *praedicare*, which means to proclaim.

From the logic point of view, a *predicate* is a relation, which of course we can also think of as a property. For example, suppose we let $p(x)$ mean "x owns a personal computer." Then p is a predicate that describes the relation (i.e., property) of owning a personal computer. Sometimes it's convenient to call $p(x)$ a predicate, although p is the actual predicate. If we replace the variable x by some definite value such as Socrates, then we obtain the proposition p(Socrates). For another example, suppose that for any two natural numbers x and y we let $q(x, y)$ mean "$x < y$." Then q is the predicate that we all know of as the "less than" relation. For example, the proposition $q(1, 5)$ is true, and the proposition $q(8, 3)$ is false.

The Existential Quantifier

Let $p(x)$ mean that x is an odd integer. Then the proposition $p(9)$ is true, and the proposition $p(20)$ is false. Similarly, the following proposition is true:

$$p(2) \vee p(3) \vee p(4) \vee p(5).$$

We can describe this proposition by saying,

"There exists an element x in the set {2, 3, 4, 5} such that $p(x)$ is true."

If we let $D = \{2, 3, 4, 5\}$, the statement can be shortened to

"There exists $x \in D$ such that $p(x)$ is true."

If we don't care where x comes from and we don't care about the meaning of $p(x)$, then we can still describe the preceding disjunction by saying:

"There exists an x such that $p(x)$."

This expression has the following symbolic representation:

$$\exists x\ p(x).$$

This expression is not a proposition because we don't have a specific set of elements over which x can vary. So $\exists x\ p(x)$ does not have a truth value. The symbol $\exists x$ is called an *existential quantifier.*

The Universal Quantifier

Now let's look at conjunctions rather than disjunctions. As before, we'll start by letting $p(x)$ mean that x is an odd integer. Suppose we have the following proposition:

$$p(1) \wedge p(3) \wedge p(5) \wedge p(7).$$

This conjunction is true and we can represent it by the following statement, where $D = \{1, 3, 5, 7\}$.

"For every x in D, $p(x)$ is true."

If we don't care where x comes from and we don't care about the meaning of $p(x)$, then we can still describe the preceding conjunction by saying:

"For every x, $p(x)$."

This expression has the following symbolic representation:

$$\forall x\ p(x).$$

This expression is not a proposition because we don't have a specific set of elements over which x can vary. So $\forall x\ p(x)$ does not have a truth value. The symbol $\forall x$ is called a *universal quantifier.*

Using Quantifiers

Let's see how the quantifiers can be used together to represent certain statements. If $p(x, y)$ is a predicate and we let the variables x and y vary over the set $D = \{0, 1\}$, then the proposition

$$[p(0, 0) \vee p(0, 1)] \wedge [p(1, 0) \vee p(1, 1)]$$

can be represented by the following quantified expression:

$$\forall x\ \exists y\ p(x, y).$$

To see this, notice that the two disjunctions can be writeen as follows:

$$p(0,0) \vee p(0,1) = \exists y\ p(0,y) \quad \text{and} \quad p(1,0) \vee p(1,1) = \exists y\ p(1,y).$$

So we can write the proposition as follows:

$$\begin{aligned}[p(0,0) \vee p(0,1)] \wedge [p(1,0) \vee p(1,1)] &= \exists y\ p(0,y) \wedge \exists y\ p(1,y)\\ &= \forall x\ \exists y\ p(x,y).\end{aligned}$$

Here's an alternative way to get the same result:

$$[p(0,0) \vee p(0,1)] \wedge [p(1,0) \vee p(1,1)] = \forall x\ [p(x,0) \vee p(x,1)] = \forall x\ \exists y\ p(x,y).$$

Now let's go the other way. We'll start with an expression containing different quantifiers and try to write it as a proposition. For example, if we use the same set of values $D = \{0,\ 1\}$, then the quantified expression

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

denotes the proposition

$$[p(0,\ 0) \wedge p(1,\ 0)] \vee [p(0,\ 1) \wedge p(1,\ 1)]\ .$$

To see this, notice that we can evaluate the quantified expression in either of two ways as follows:

$$\begin{aligned}\exists y\ \forall x\ p(x,y) &= \forall x\ p(x,0) \vee \forall x\ p(x,1) = [p(0,0) \wedge p(1,0)] \vee [p(0,1) \wedge p(1,1)].\\ \exists y\ \forall x\ p(x,y) &= \exists y\ [p(0,y) \wedge p(1,y)] = [p(0,0) \wedge p(1,0)] \vee [p(0,1) \wedge p(1,1)].\end{aligned}$$

Of course, not every expression containing quantifiers results in a proposition. For example, if $D = \{0,\ 1\}$, then the expression $\forall x\ p(x,\ y)$ can be written as follows:

$$\forall x\ p(x,\ y) = p(0,\ y) \wedge p(1,\ y).$$

To obtain a proposition, each variable of the expression must be quantified or assigned some value in D. We'll discuss this shortly when we talk about semantics.

Now let's look at two examples, which will introduce us to the important process of formalizing English sentences with quantifiers.

example 7.1 Formalizing Sentences

We'll formalize the three sentences about Socrates that we listed at the beginning of the section. Over the domain of all people, let $C(x)$ mean that x is a computer science major, and let $P(x)$ mean that x owns a personal computer. Then the sentence "All computer science majors own a personal computer" can be formalized as

$$\forall x\ (C(x) \rightarrow P(x)).$$

The sentence "Socrates does not own a personal computer" becomes

$$\neg\, P(\text{Socrates}).$$

The sentence "Socrates is not a computer science major" becomes

$$\neg\, C(\text{Socrates}).$$

end example

example

7.2 Formalizing Sentences

Suppose we consider the following two elementary facts about the set ℕ of natural numbers.

1. Every natural number has a successor.

2. There is no natural number whose successor is 0.

Let's formalize these sentences. We'll begin by writing down a semiformal version of the first sentence:

For each $x \in \mathbb{N}$ there exists $y \in \mathbb{N}$ such that y is the successor of x.

If we let $s(x, y)$ mean that y is the successor of x, then the formal version of the sentence can be written as follows:

$$\forall x\ \exists y\ s(x, y).$$

Now let's look at the second sentence. It can be written in a semiformal version as follows:

There does not exist $x \in \mathbb{N}$ such that the successor of x is 0.

The formal version of this sentence is the following sentence, where $a = 0$:

$$\neg\, \exists x\ s(x, a).$$

end example

These notions of quantification belong to a logic called *first-order predicate calculus*. The term "first-order" refers to the fact that quantifiers can quantify only variables that occur in predicates. In Chapter 8 we'll discuss "higher-order" logics in which quantifiers can quantify additional things. To discuss first-order predicate calculus, we need to give a precise description of its well-formed formulas and their meanings. That's the task of this section.

7.1.2 Well-Formed Formulas

To give a precise description of a first-order predicate calculus, we need an alphabet of symbols. For this discussion we'll use several kinds of letters and symbols, described as follows:

Individual variables:	x, y, z
Individual constants:	a, b, c
Function constants:	f, g, h
Predicate constants:	p, q, r
Connective symbols:	$\neg$, $\rightarrow$, $\wedge$, $\vee$
Quantifier symbols:	$\exists$, $\forall$
Punctuation symbols:	(,)

From time to time we will use other letters, or strings of letters, to denote variables or constants. We'll also allow letters to be subscripted. The number of arguments for a predicate or function will normally be clear from the context. A predicate with no arguments is considered to be a proposition.

A *term* is either a variable, a constant, or a function applied to arguments that are terms. For example, x, a, and $f(x, g(b))$ are terms. An *atomic formula* (or simply *atom*) is a predicate applied to arguments that are terms. For example, $p(x, a)$ and $q(y, f(c))$ are atoms.

We can define the wffs—the well-formed formulas—of the first-order predicate calculus inductively as follows:

Definition of a Wff (Well-Formed Formula)

1. Any atom is a wff.
2. If W and V are wffs and x is a variable, then the following expressions are also wffs:

$$(W),\ \neg\ W,\ W \vee V,\ W \wedge V,\ W \rightarrow V,\ \exists x\ W, \text{ and } \forall x\ W.$$

To write formulas without too many parentheses and still maintain a unique meaning, we'll agree that the quantifiers have the same precedence as the negation symbol. We'll continue to use the same hierarchy of precedence for the operators $\neg$, $\wedge$, $\vee$, and $\rightarrow$. Therefore, the hierarchy of precedence now looks like the following:

$\neg$, $\exists x$, $\forall y$	(highest, do first)
$\wedge$	
$\vee$	
$\rightarrow$	(lowest, do last)

If any of the quantifiers or the negation symbol appear next to each other, then the rightmost symbol is grouped with the smallest wff to its right. Here are a few wffs in both unparenthesized form and parenthesized form:

Unparenthesized Form	*Parenthesized Form*
$\forall x \neg \exists y \forall z\, p(x, y, z)$	$\forall x (\neg (\exists y (\forall z\, p(x, y, z))))$.
$\exists x\, p(x) \vee q(x)$	$(\exists x\, p(x)) \vee q(x)$.
$\forall x\, p(x) \to q(x)$	$(\forall x\, p(x)) \to q(x)$.
$\exists x \neg p(x, y) \to q(x) \wedge r(y)$	$(\exists x(\neg p(x, y))) \to (q(x) \wedge r(y))$.
$\exists x\, p(x) \to \forall x\, q(x) \vee p(x) \wedge r(x)$	$(\exists x\, p(x)) \to ((\forall x\, q(x)) \vee (p(x) \wedge r(x)))$.

Scope, Bound, and Free

Now let's discuss the relationship between the quantifiers and the variables that appear in a wff. When a quantifier occurs in a wff, it influences some occurrences of the quantified variable. The extent of this influence is called the scope of the quantifier, which we define as follows:

Definition of Scope

In the wff $\exists x\ W$, W is the *scope* of the quantifier $\exists x$.

In the wff $\forall x\ W$, W is the *scope* of the quantifier $\forall x$.

For example, the scope of $\exists x$ in the wff

$$\exists x\, p(x, y) \to q(x)$$

is $p(x, y)$ because the parenthesized version of the wff is $(\exists x\, p(x, y)) \to q(x)$. On the other hand, the scope of $\exists x$ in the wff

$$\exists x\ (p(x, y) \to q(x))$$

is the conditional $p(x, y) \to q(x)$. Now let's classify the occurrences of variables that occur in a wff.

Bound and Free Variables

An occurrence of a variable x in a wff is said to be *bound* if it lies within the scope of either $\exists x$ or $\forall x$ or if it is the quantifier variable x itself. Otherwise, an occurrence of x is said to be *free* in the wff.

For example, consider the following wff:

$$\exists x\ p(x,\ y) \to q(x).$$

The first two occurrences of x are bound because the scope of $\exists x$ is $p(x,\ y)$. The only occurrence of y is free, and the third occurrence of x is free.

So every occurrence of a variable in a wff can be classified as either bound or free, and this classification is determined by the scope of the quantifiers in the wff. Now we're in a position to discuss the meaning of wffs.

7.1.3 Semantics and Interpretations

Up to this point a wff is just a string of symbols with no meaning attached. For a wff to have a meaning, we must give an interpretation to its symbols so that the wff can be read as a statement that is true or false. For example, suppose we let $p(x)$ mean "x is an even integer" and we let x be the number 236. With this interpretation, $p(x)$ becomes the statement "236 is an even integer," which is true.

As another example, let's give an interpretation to the wff

$$\forall x\ \exists y\ s(x,\ y).$$

We'll let $s(x,\ y)$ mean that the successor of x is y, where the variables x and y take values from the set of natural numbers $\mathbb{N}$. With this interpretation the wff becomes the statement "For every natural number x there exists a natural number y such that the successor of x is y," which is true.

Before we proceed any further, we need to make a precise definition of the idea of an interpretation. Here's the definition in all its detail.

Definition of Interpretation

An *interpretation* for a wff consists of a nonempty set D, which is called the *domain* of the interpretation, together with an assignment that associates the symbols of the wff to values in D as follows:

1. Each predicate letter is assigned to a relation over D. A predicate with no arguments is a proposition and must be assigned a truth value.
2. Each function letter is assigned to a function over D.
3. Each free variable is assigned to a value in D. All free occurrences of a variable x are assigned to the same value in D.
4. Each constant is assigned to a value in D. All occurrences of the same constant are assigned to the same value in D.

Substitutions for Free Variables

Suppose W is a wff, x is a free variable in W, and t is a term. Then the wff obtained from W by replacing all free occurrences of x by t is denoted by the expression

$$W(x/t).$$

The expression x/t is called a *substitution.* We should observe that if x is not free in W, then x/t does not change W. In other words:

If x is not free in W, then we have $W(x/t) = W$.

For example, let $W = \forall x\ p(x, y)$. We'll calculate $W(x/t)$ and $W(y/t)$.

$$W(x/t) = \forall x\ p(x, y) = W \quad \text{and} \quad W(y/t) = \forall x\ p(x, t).$$

example 7.3 Substituting for a Variable

Let $W = p(x, y) \vee \exists y\ q(x, y)$. Notice that x occurs free twice in W, so for any term t we can apply the substitution x/t to W to obtain

$$W(x/t) = p(t, y) \vee \exists y\ q(t, y).$$

Since t can be any term, we can come up with many different results. For example,

$$\begin{aligned} W(x/a) &= p(a, y) \vee \exists y\ q(a, y), \\ W(x, y) &= p(y, y) \vee \exists y\ q(y, y), \\ W(x/z) &= p(z, y) \vee \exists y\ q(z, y), \\ W(x/f(x, y, z)) &= p(f(x, y, z), y) \vee \exists y\ q(f(x, y, z), y). \end{aligned}$$

Notice that y occurs free once in W, so for any term t we can apply the substitution y/t to W to obtain

$$W(y/t) = p(x, t) \vee \exists y q(x, y).$$

We can also apply one substitution and then another. For example,

$$W(x/a)(y/b) = (p(a, y) \vee \exists y\ q(a, y))(y/b) = p(a, b) \vee \exists y\ q(a, y).$$

end example

Let's record some simple yet useful facts about substitutions, all of which follow directly from the definition.

Properties of Substitutions **(7.1)**

1. x/t distributes over the connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$. For example,

$$\begin{aligned}(\neg A)(x/t) &= \neg A(x/t).\\ (A \wedge B)(x/t) &= A(x/t) \wedge B(x/t).\\ (A \vee B)(x/t) &= A(x/t) \vee B(x/t).\\ (A \rightarrow B)(x/t) &= A(x/t) \rightarrow B(x/t).\end{aligned}$$

2. If $x \neq y$, then x/t distributes over $\forall y$ and $\exists y$. For example,

$$\begin{aligned}(\forall y W)(x/t) &= \forall y(W(x/t)).\\ (\exists y W)(x/t) &= \exists y(W(x/t)).\end{aligned}$$

3. If x is not free in W, then $W(x/t) = W$. For example,

$$\begin{aligned}(\forall x W)(x/t) &= \forall x W.\\ (\exists x W)(x/t) &= \exists x W.\end{aligned}$$

The Meaning of a Wff

Now we have all the ingredients to define the *meaning*, or *semantics*, of wffs in first-order predicate calculus. Informally, the meaning of a wff with respect to an interpretation will be either true or false depending on how the interpretation is defined. But with quantifiers in the picture, we need to make a precise definition of how to find the truth of a wff. Here's that definition.

The Meaning of a Wff

The meaning of a wff with respect to an interpretation with domain D is the truth value obtained by applying the following rules:

1. If the wff has no quantifiers, then its meaning is the truth value of the proposition obtained by applying the interpretation to the wff.
2. If the wff contains $\forall x\ W$, then the meaning of $\forall x\ W$ is true if $W(x/d)$ is true for every $d \in D$. Otherwise, the meaning of $\forall x\ W$ is false.
3. If the wff contains $\exists x\ W$, then the meaning of $\exists x\ W$ is true if $W(x/d)$ is true for some $d \in D$. Otherwise, the meaning of $\exists x\ W$ is false.

When a wff W is true (or false) with respect to an interpretation I, we'll often say that W *is true* (*or false*) *for* I.

So the meaning of a wff without quantifiers is just the truth value of the proposition obtained by applying the interpretation to all the free variables and constants in the wff. The meaning of a wff containing quantifiers can be computed by recursively applying the definition. We'll do an example.

example 7.4 Applying an Interpretation

Let's consider a wff of the form $\forall x \; \exists y \; W$, where W does not contain any further quantifiers. Suppose we have an interpretation with domain D. Then the meaning of $\forall x \; \exists y \; W$ is true if the meaning of $(\exists y \; W)(x/d)$ is true for every $d \in D$. Since $x \neq y$, the properties of substitution tell us that

$$(\exists y \; W)(x/d) = \exists y \; (W(x/d)).$$

So for each $d \in D$ we must find the meaning of $\exists y \; (W(x/d))$. Now the meaning of $\exists y \; (W(x/d))$ is true if there is some element $e \in D$ such that $W(x/d)(y/e)$ is true. Since our assumption is that W does not contain any further quantifiers, the meaning of $W(x/d)(y/e)$ is the truth value of the proposition obtained by applying the interpretation to any remaining free variables and constants.

end example

Notation

Whenever we want to emphasize the fact that a wff W might contain a free variable x, we'll represent W by the expression

$$W(x).$$

When this is the case, we often write $W(t)$ to denote the wff $W(x/t)$.

Now let's look at some examples with concrete interpretations to see that things are not as complicated as they might seem.

example 7.5 Converting to English

Consider the wff $\forall x \; \exists y \; s(x, y)$ with the interpretation having domain $D = \mathbb{N}$, and $s(x, y)$ means that the successor of x is y. The interpreted wff can be restated as "Every natural number has a successor."

end example

example

7.6 A Parental Relationship

Let's consider the "isFatherOf" predicate, where isFatherOf(x, y) means "x is the father of y." The domain of our interpretation is the set of all people now living or who have lived. Assume also that Jim is the father of Andy. The following wffs are given along with their interpreted values:

$$\begin{aligned}
&\text{isFatherOf(Jim, Andy)} = \text{true},\\
&\exists x\ \text{isFatherOf}(x, \text{Andy}) = \text{true},\\
&\forall x\ \text{isFatherOf}(x, \text{Andy}) = \text{false},\\
&\forall x\ \exists y\ \text{isFatherOf}(x, y) = \text{false},\\
&\forall y\ \exists x\ \text{isFatherOf}(x, y) = \text{true},\\
&\exists x\ \forall y\ \text{isFatherOf}(x, y) = \text{false},\\
&\exists y\ \forall x\ \text{isFatherOf}(x, y) = \text{false},\\
&\exists x\ \exists y\ \text{isFatherOf}(x, y) = \text{true},\\
&\exists y\ \exists x\ \text{isFatherOf}(x, y) = \text{true},\\
&\forall x\ \forall y\ \text{isFatherOf}(x, y) = \text{false},\\
&\forall y\ \forall x\ \text{isFatherOf}(x, y) = \text{false}.
\end{aligned}$$

end example

example

7.7 Different Interpretations

Let's look at some different interpretations for $W = \exists x\ \forall y\ (p(y) \to q(x, y))$. For each of the following interpretations we'll let $q(x, y)$ denote the equality relation "$x = y$."

a. Let the domain $D = \{a\}$, and let $p(a)$ = true. Then W is true.

b. Let the domain $D = \{a\}$, and let $p(a)$ = false. Then W is true.

c. Let the domain $D = \{a, b\}$, and let $p(a) = p(b)$ = true. Then W is false.

d. Notice that W is true for any domain D for which $p(d)$ = true for at most one element $d \in D$.

end example

example

7.8 A Function Symbol

Let $W = \forall x\ (p(f(x, x), x) \to p(x, y))$. One interpretation for W can be made as follows: Let $\mathbb{N}$ be the domain, let p be equality, let $y = 0$, and let f be the function defined by $f(a, b) = (a + b) \bmod 3$. With this interpretation, W can be written in more familiar notation as follows:

$$\forall x\ ((2x \bmod 3 = x) \to x = 0).$$

A bit of checking will convince us that W is true with respect to this interpretation.

end example

example **7.9 A Function Symbol**

Let $W = \forall x\ (p(f(x,\ x),\ x) \rightarrow p(x,\ y))$. Let $D = \{a,\ b\}$ be the domain of an interpretation such that $f(a,\ a) = a$, $f(b,\ b) = b$, p is equality, and $y = a$. Then W is false with respect to this interpretation.

end example

Models and Countermodels

An interpretation for a wff W is called a *model* for W if W is true with respect to the interpretation. Otherwise, the interpretation is a *countermodel* for W. The previous two examples gave a model and a countermodel, respectively, for the wff

$$W = \forall x\ (p(f(x,\ x),\ x) \rightarrow p(x,\ y)).$$

7.1.4 Validity

Can any wff be true for every possible interpretation? Although it may seem unlikely, this property holds for many wffs. The property is important enough to introduce some terminology. A wff is *valid* if it's true for all possible interpretations. So a wff is valid if every interpretation is a model. Otherwise, the wff is *invalid*. A wff is *unsatisfiable* if it's false for all possible interpretations. So a wff is unsatisfiable if all of its interpretations are countermodels. Otherwise, it is *satisfiable*. From these definitions we see that every wff satisfies exactly one of the following pairs of properties:

valid and satisfiable,

satisfiable and invalid,

unsatisfiable and invalid.

In the propositional calculus the words *tautology*, *contingency*, and *contradiction* correspond, respectively, to the preceding three pairs of properties.

example **7.10 A Satisfiable and Invalid Wff**

The wff $\exists x\ \forall y\ (p(y) \rightarrow q(x,\ y))$ is satisfiable and invalid. To see that the wff is satisfiable, notice that the wff is true with respect to the following interpretation: The domain is the singleton $\{3\}$, and we define $p(3) =$ true and $q(3,\ 3) =$ true.

To see that the wff is invalid, notice that it is false with respect to the following interpretation: The domain is still the singleton $\{3\}$, but now we define $p(3) =$ true and $q(3, 3) =$ false.

end example

Proving Validity

In the propositional calculus we can use truth tables to decide whether any propositional wff is a tautology. But how can we show that a wff of the predicate calculus is valid? We can't check the infinitely many interpretations of the wff to see whether each one is a model. So we are forced to use some kind of reasoning to show that a wff is valid. Here are two strategies to prove validity.

Direct approach: If the wff has the form $A \to B$, then assume that there is an arbitrary interpretation for $A \to B$ that is a model for A. Show that the interpretation is a model for B. This proves that any interpretation for $A \to B$ is a model for $A \to B$. So $A \to B$ is valid.

Indirect approach: Assume that the wff is invalid, and try to obtain a contradiction. Start by assuming the existence of a countermodel for the wff. Then try to argue toward a contradiction of some kind. For example, if the wff has the form $A \to B$, then a countermodel for $A \to B$ makes A true and B false. This information should be used to find a contradiction.

We'll demonstrate these proof strategies in the next example. But first we'll list a few valid conditionals to have something to talk about.

Some Valid Conditionals **(7.2)**

a. $\forall x\ A(x) \to \exists x\ A(x)$.

b. $\exists x\ (A(x) \wedge B(x)) \to \exists x\ A(x) \wedge \exists x\ B(x)$.

c. $\forall x\ A(x) \vee \forall x\ B(x) \to \forall x\ (A(x) \vee B(x))$.

d. $\forall x\ (A(x) \to B(x)) \to (\forall x\ A(x) \to \forall x\ B(x))$.

e. $\exists y\ \forall x\ P(x, y) \to \forall\ x\ \exists y\ P(x, y)$.

We should note that the converses of these wffs are invalid. We'll leave this to the exercises. In the following example we'll use the direct approach and the indirect approach to prove the validity of (7.2e). The proofs of (7.2a–7.2d) are left as exercises.

example

7.11 Proving Validity

Let W denote the following wff:

$$\exists y\ \forall x\ P(x,\ y) \to \forall x\ \exists y\ P(x,\ y).$$

We'll give two proofs to show that W is valid—one direct and one indirect. In both proofs we'll let A be the antecedent and B be the consequent of W.

Direct approach: Let M be an interpretation with domain D for W such that M is a model for A. Then there is an element $d \in D$ such that $\forall x\ P(x,\ d)$ is true. Therefore, $P(e,\ d)$ is true for all $e \in D$. This says that M is also a model for B. Therefore, W is valid. QED.

Indirect approach: Assume that W is invalid. Then it has a countermodel with domain D that makes A true and B false. Therefore, there is an element $d \in D$ such that the wff $\exists y\ P(d,\ y)$ is false. Thus $P(d,\ e)$ is false for all $e \in D$. Now we are assuming that A is true. Therefore, there is an element $c \in D$ such that $\forall x$ $P(x,\ c)$ is true. In other words, $P(b,\ c)$ is true for all $b \in D$. In particular, this says that $P(d,\ c)$ is true. But this contradicts the fact that $P(d,\ e)$ is false for all elements $e \in D$. Therefore, W is valid. QED.

end example

Closures

There are two interesting transformations that we can apply to any wff containing free variables. One is to universally quantify each free variable, and the other is to existentially quantify each free variable. It seems reasonable to expect that these transformations will change the meaning of the original wff, as the following examples show:

$p(x) \wedge \neg\ p(y)$ is satisfiable, but $\forall x\ \forall y\ (p(x) \wedge \neg\ p(y))$ is unsatisfiable.

$p(x) \to p(y)$ is invalid, but $\exists x\ \exists y\ (p(x) \to p(y))$ is valid.

The interesting thing about the process is that validity is preserved if we universally quantify the free variables and unsatisfiability is preserved if we existentially quantify the free variables. To make this more precise, we need a little terminology.

Suppose W is a wff with free variables $x_1, \ldots, x_n$. The *universal closure* of W is the wff

$$\forall x_1 \cdots \forall x_n\ W.$$

The *existential closure* of W is the wff

$$\exists x_1 \cdots \exists x_n\ W.$$

For example, suppose $W = \forall x\, p(x, y)$. W has y as its only free variable. So the universal closure of W is

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

and the existential closure of W is

$$\exists y\ \forall x\ p(x, y).$$

As we have seen, the meaning of a wff may change by taking either of the closures. But there are two properties that don't change, and we'll state them for the record as follows:

Closure Properties **(7.3)**

1. A wff is valid if and only if its universal closure is valid.
2. A wff is unsatisfiable if and only if its existential closure is unsatisfiable.

Proof: We'll prove part (1) first. To start things off we'll show that if x is a free variable in a wff W, then

$$W \text{ is valid if and only if } \forall x\ W \text{ is valid.}$$

Suppose that W is valid. Let I be an interpretation with domain D for the wff $\forall x\ W$. If I is not a model for $\forall x\ W$, then there is some element $d \in D$ such that $W(x/d)$ is false for I. This being the case, we can define an interpretation J for W by letting J be I, with the free variable x assigned to the element $d \in D$. Since W is valid, it follows that $W(x/d)$ is true for J. But $W(x/d)$ with respect to J is the same as $W(x/d)$ with respect to I, which is false. This contradiction shows that I is a model for $\forall x\ W$. Therefore, $\forall x\ W$ is valid.

Suppose $\forall x\ W$ is valid. Let I be an interpretation with domain D for W, where x is assigned the value $d \in D$. Now define an interpretation J for $\forall x\ W$ by letting J be obtained from I by removing the assignment of x to d. Then J is an interpretation for the valid wff $\forall x\ W$. So $W(x/e)$ is true for J for all elements $e \in D$. In particular, $W(x/d)$ is true for J, and thus also for I. Therefore, I is a model for W. Therefore, W is valid.

The preceding two paragraphs tell us that if x is free in W, then W is valid if and only if $\forall x\ W$ is valid. The proof now follows by induction on the number n of free variables in a wff W. If $n = 0$, then W does not have any free variables, so W is its own universal closure. So assume that $n > 0$ and assume that part (1) is true for any wff with k free variables, where $k < n$. If x is a free variable, then $\forall x\ W$ contains $n - 1$ free variables, and it follows by induction that $\forall x\ W$ is valid if and only if its universal closure is valid. But the universal closure of $\forall x\ W$ is the same as the universal closure of W. So it follows that W is valid if and only if the universal closure of W is valid. This proves part (1).

Part (2) is easy. We'll use part (1) and a sequence of iff statements. W is unsatisfiable iff $\neg W$ is valid iff the universal closure of $\neg W$ is valid iff the negation of the existential closure of W is valid iff the existential closure of W is unsatisfiable. QED.

7.1.5 The Validity Problem

We'll end this section with a short discussion about deciding the validity of wffs. First we need to introduce the general notion of decidability. Any problem that can be stated as a question with a yes or no answer is called a *decision problem*. Practically every problem can be stated as a decision problem, perhaps after some work. A decision problem is called *decidable* if there is an algorithm that halts with the answer to the problem. Otherwise, the problem is called *undecidable*. A decision problem is called *partially decidable* if there is an algorithm that halts with the answer yes if the problem has a yes answer but may not halt if the problem has a no answer. The words *solvable*, *unsolvable*, and *partially solvable* are also used to mean decidable, undecidable, and partially decidable, respectively.

Now let's get back to logic. The *validity problem* for a formal theory can be stated as follows:

Given a wff, is it valid?

The validity problem for the propositional calculus can be stated as follows: Given a wff, is it a tautology? This problem is decidable by Quine's method. Another algorithm would be to build a truth table for the wff and then check it.

Although the validity problem for the first-order predicate calculus is undecidable, it is partially decidable. There are two partial decision procedures for the first-order predicate calculus that are of interest: *natural deduction* (due to Gentzen [1935]) and *resolution* (due to Robinson [1965]). Natural deduction is a formal reasoning system that models the natural way we reason about the validity of wffs by using inference rules, as we did in Chapter 6 and as we'll discuss in Section 7.3. Resolution is a mechanical way to reason, which is not easily adaptable to people. It is, however, adaptable to machines. Resolution is an important ingredient in logic programming and automatic reasoning, which we'll discuss in Chapter 9.

Exercises

Quantified Expressions

1. Write down the proposition denoted by each of the following expressions, where the variables take values in the domain $\{0, 1\}$.

   a. $\exists x \, \forall y \; p(x, y)$.

   b. $\forall y \, \exists x \; p(x, y)$.

2. Write down a quantified expression over some domain to denote each of the following propositions or predicates.

   a. $q(0) \wedge q(1)$.
   b. $q(0) \vee q(1)$.
   c. $p(x, 0) \wedge p(x, 1)$.
   d. $p(0, x) \vee p(1, x)$.
   e. $p(1) \vee p(3) \vee p(5) \vee \cdots$.
   f. $p(2) \wedge p(4) \wedge p(6) \wedge \cdots$.

Syntax, Scope, Bound, and Free

3. Explain why each of the following expressions is a wff.

   a. $\exists x\ p(x) \rightarrow \forall x\ p(x)$.
   b. $\exists x \forall y\ (p(y) \rightarrow q(f(x), y))$.

4. Explain why the expression $\forall\ y\ (p(y) \rightarrow q(f(x), p(x)))$ is not a wff.

5. For each of the following wffs, label each occurrence of the variables as either bound or free.

   a. $p(x, y) \vee (\forall y\ q(y) \rightarrow \exists x\ r(x, y))$.
   b. $\forall y\ q(y) \wedge \neg\ p(x, y)$.
   c. $\neg\ q(x, y) \vee \exists x\ p(x, y)$.

6. Write down a single wff containing three variables x, y, and z, with the following properties: x occurs twice as a bound variable; y occurs once as a free variable; z occurs three times, once as a free variable and twice as a bound variable.

Interpretations

7. Let $B(x)$ mean x is a bird, let $W(x)$ mean x is a worm, and let $E(x, y)$ mean x eats y. Find an English sentence to describe each of the following statements.

   a. $\forall x\ \forall\ y\ (B(x) \wedge W(y) \rightarrow E(x, y))$.
   b. $\forall x\ \forall\ y\ (E(x, y) \rightarrow B(x) \wedge W(y))$.

8. Let $p(x)$ mean that x is a person, let $c(x)$ mean that x is a chocolate bar, and let $e(x, y)$ mean that x eats y. For each of the following wffs, write down an English sentence that reflects the interpretation of the wff.

   a. $\exists x\ (p(\mathrm{x}) \wedge \forall y\ (c(y) \rightarrow e(x, y)))$.
   b. $\forall y\ (c(y) \wedge \exists x\ (p(\mathrm{x}) \wedge e(x, y)))$.

9. Let $e(x, y)$ mean that $x = y$, let $p(x, y)$ mean that $x < y$, and let $d(x, y)$ mean that x divides y. For each of the following statements about the natural numbers, find a formal quantified expression.

a. Every natural number other than 0 has a predecessor.
b. Any two nonzero natural numbers have a common divisor.

10. Given the wff $W = \exists x\, p(x) \rightarrow \forall\, x\, p(x)$.

a. Find all possible interpretations of W over the domain $D = \{a\}$. Also give the truth value of W over each of the interpretations.
b. Find all possible interpretations of W over the domain $D = \{a, b\}$. Also give the truth value of W over each of the interpretations.

11. Find a model for each of the following wffs.

a. $p(c) \wedge \exists x \neg p(x)$.
b. $\exists x\, p(x) \rightarrow \forall x\, p(x)$.
c. $\exists y\, \forall x\, p(x, y) \rightarrow \forall x\, \exists y\, p(x, y)$.
d. $\forall x\, \exists y\, p(x, y) \rightarrow \exists y\, \forall x\, p(x, y)$.
e. $\forall x\, (p(x, f(x)) \rightarrow p(x, y))$.

12. Find a countermodel for each of the following wffs.

a. $p(c) \wedge \exists x \neg p(x)$.
b. $\exists x\, p(x) \rightarrow \forall x\, p(x)$.
c. $\forall x\, (p(x) \vee q(x)) \rightarrow \forall x\, p(x) \vee \forall x\, q(x)$.
d. $\exists x\, p(x) \wedge \exists x\, q(x) \rightarrow \exists x\, (p(x) \wedge q(x))$.
e. $\forall x\, \exists y\, p(x, y) \rightarrow \exists y\, \forall x\, p(x, y)$.
f. $\forall x\, (p(x, f(x)) \rightarrow p(x, y))$.
g. $(\forall x\, p(x) \rightarrow \forall x\, q(x)) \rightarrow \forall x\, (p(x) \rightarrow q(x))$.

Validity

13. Given the wff $W = \forall x\, \forall y\, (p(x) \rightarrow p(y))$.

a. Show that W is true for any interpretation whose domain is a singleton.
b. Show that W is not valid.

14. Given the wff $W = \forall x\, p(x, x) \rightarrow \forall x\, \forall y\, \forall z\, (p(x, y) \vee p(x, z) \vee p(y, z))$.

a. Show that W is true for any interpretation whose domain is a singleton.
b. Show that W is true for any interpretation whose domain has two elements.
c. Show that W is not valid.

15. Find an example of a wff that is true for any interpretation having a domain with three or fewer elements but is not valid. *Hint:* Look at the structure of the wff in Exercise 14.

16. Prove that each of the following wffs is valid. *Hint:* Either show that every interpretation is a model or assume that the wff is invalid and find a contradiction.

a. $\forall x\ (p(x) \to p(x))$.
b. $p(c) \to \exists x\, p(x)$.
c. $\forall x\, p(x) \to \exists x\, p(x)$.
d. $\exists x\ (A(x) \wedge B(x)) \to \exists x\, A(x) \wedge \exists x\, B(x)$.
e. $\forall x\, A(x) \vee \forall x\, B(x) \to \forall x\ (A(x) \vee B(x))$.
f. $\forall x\ (A(x) \to B(x)) \to (\exists x\, A(x) \to \exists x\, B(x))$.
g. $\forall x\ (A(x) \to B(x)) \to (\forall x\, A(x) \to \exists x\, B(x))$.
h. $\forall x\ (A(x) \to B(x)) \to (\forall x\, A(x) \to \forall x\, B(x))$.

17. Prove that each of the following wffs is unsatisfiable. *Hint:* Either show that every interpretation is a countermodel or assume that the wff is satisfiable and find a contradiction.

a. $p(c) \wedge \neg\, p(c)$.
b. $\exists x\ (p(x) \wedge \neg\, p(x))$.
c. $\exists x\, \forall y\ (p(x, y) \wedge \neg\, p(x, y))$.

Further Thoughts

18. For a wff W, let $c(W)$ denote the wff obtained from W by replacing the free variables of W by distinct constants. Prove that W has a model if and only if $c(W)$ has a model.

19. Prove that any wff of the form $A \to B$ is valid if and only if whenever A is valid, then B is valid.

20. Prove part (2) of (7.3) by using a proof similar to that of part (1). A wff is unsatisfiable if and only if its existential closure is unsatisfiable.

7.2 Equivalent Formulas

In this section we'll discuss the important notion of equivalence for wffs of the first-order predicate calculus.

7.2.1 Equivalence

Two wffs A and B are said to be *equivalent* if they both have the same truth value with respect to every interpretation of both A and B. By an interpretation of both A and B, we mean that all free variables, constants, functions, and predicates that occur in either A or B are interpreted with respect to a single domain. If A and B are equivalent, then we write

$$A \equiv B.$$

We should note that any two valid wffs are equivalent because they are both true for any interpretation. Similarly, any two unsatisfiable wffs are equivalent because they are both false for any interpretation. The definition of equivalence also allows us to make the following useful formulation in terms of conditionals and validity.

Equivalence

$$\begin{aligned} A \equiv B \quad &\text{if and only if} \quad (A \to B) \wedge (B \to A) \text{ is valid} \\ &\text{if and only if} \quad A \to B \text{ and } B \to A \text{ are both valid.} \end{aligned}$$

Instances of Propositional Wffs

To start things off, let's see how propositional equivalences give rise to predicate calculus equivalences. A wff W is an *instance* of a propositional wff V if W is obtained from V by replacing each propositional variable of V by a wff, where all occurrences of each propositional variable in V are replaced by the same wff. For example, the wff

$$\forall x\, p(x) \to \forall x\, p(x) \vee q(x)$$

is an instance of $P \to P \vee Q$ because Q is replaced by $q(x)$ and both occurrences of P are replaced by $\forall x\, p(x)$.

If W is an instance of a propositional wff V, then the truth value of W for any interpretation can be obtained by assigning truth values to the propositional variables of V. For example, suppose we define an interpretation with domain $D = \{a, b\}$ and we set $p(a) = p(b) =$ true and $q(a) = q(b) =$ false. For this interpretation, the truth value of the wff $\forall x\, p(x) \to \forall x\, p(x) \vee q(x)$ is the same as the truth value of the propositional wff $P \to P \vee Q$, where $P =$ true and $Q =$ false.

So we can say that two wffs are equivalent if they are instances of two equivalent propositional wffs, where both instances are obtained by using the same replacement of propositional variables. For example, we have

$$\forall x\, p(x) \to q(x) \equiv \neg \forall x\, p(x) \vee q(x)$$

because the left and right sides are instances of the left and right sides of the propositional equivalence $P \to Q \equiv \neg P \vee Q$, where both occurrences of P are replaced by $\forall x\, p(x)$ and both occurrences of Q are replaced by $q(x)$. We'll state the result again for emphasis:

Two wffs are equivalent whenever they are instances of two equivalent propositional wffs, where both instances are obtained by using the same replacement of propositional variables.

Equivalences involving Quantifiers

Let's see whether we can find some more equivalences to make our logical life easier. We'll start by listing equivalences that relate the two quantifiers by negation. For any wff W we have the following two equivalences.

Quantifiers and Negation **(7.4)**

$$\neg (\forall x\ W) \equiv \exists x \neg W \quad \text{and} \quad \neg (\exists x\ W) \equiv \forall x \neg W.$$

It's easy to believe that these two equivalences are true. For example, we can illustrate the equivalence $\neg (\forall x\ W) \equiv \exists x \neg W$ by observing that the negation of the statement "Something is true for all possible cases" has the same meaning as the statement "There is some case for which the something is false." Similarly, we can illustrate the equivalence $\neg (\exists x\ W) \equiv \forall x \neg W$ by observing that the negation of the statement "There is some case for which something is true" has the same meaning as the statement "Every case of the something is false."

Another way to demonstrate these equivalences is to use De Morgan's laws. For example, let $W = p(x)$ and suppose that we have an interpretation with domain $D = \{0, 1, 2, 3\}$. Then no matter what values we assign to p, we can apply De Morgan's laws to obtain the following propositional equivalence:

$$\begin{aligned} \neg (\forall x\ p(x)) &\equiv \neg (p(0) \wedge p(1) \wedge p(2) \wedge p(3)) \\ &\equiv \neg p(0) \vee \neg p(1) \vee \neg p(2) \vee \neg p(3) \\ &\equiv \exists x \neg p(x). \end{aligned}$$

We also get the following equivalence:

$$\begin{aligned} \neg (\exists x\ p(x)) &\equiv \neg (p(0) \vee p(1) \vee p(2) \vee p(3)) \\ &\equiv \neg p(0) \wedge \neg p(1) \wedge \neg p(2) \wedge \neg p(3) \\ &\equiv \forall x \neg p(x). \end{aligned}$$

These examples are nice, but they don't prove (7.4). Let's give an actual proof, using validity, of the equivalences (7.4). We'll prove the first equivalence, $\neg (\forall x\ W) \equiv \exists x \neg W$, and then use it to prove the second equivalence.

Proof: Let I be an interpretation with domain D for the wffs $\neg (\forall x\ W)$ and $\exists x \neg W$. We want to show that I is a model for one of the wffs if and only if I is a model for the other wff. The following equivalent statements do the job:

$$\begin{aligned} I \text{ is a model for } \neg(\forall x W) \quad & \text{iff} \quad \neg(\forall x W) \text{ is true for } I \\ & \text{iff} \quad \forall x W \text{ is false for } I \\ & \text{iff} \quad W(x/d) \text{ is false for some } d \in D \\ & \text{iff} \quad \neg W(x/d) \text{ is true for some } d \in D \\ & \text{iff} \quad \exists x \neg W \text{ is true for } I \\ & \text{iff} \quad I \text{ is a model for } \exists x \neg W. \end{aligned}$$

This proves the equivalence $\neg (\forall x\ W) \equiv \exists x \neg W$. Now, since W is arbitrary, we can replace W by the wff $\neg W$ to obtain the following equivalence:

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

Now take the negation of both sides of this equivalence, and simplify the double negations to obtain the second equivalence of (7.4):

$$\forall x \neg W \equiv \neg (\exists x\ W). \text{ QED.}$$

Now let's look at two equivalences that allow us to interchange universal quantifiers if they are next to each other; the same holds for existential quantifiers.

Interchanging Quantifers of the Same Type **(7.5)**

$$\forall x\, \forall y\ W \equiv \forall y\, \forall\, x\ W \quad \text{and} \quad \exists x\, \exists y\ W \equiv \exists y\, \exists x\ W.$$

Again, this is easy to believe. For example, suppose that $W = p(x,\ y)$ and we have an interpretation with domain $D = \{0,\ 1\}$. Then we have the following equivalences.

$$\begin{aligned} \forall x\ \forall y\ p(x,y) & \equiv \forall y\ p(0,y) \wedge \forall y\ p(1,y) \\ & \equiv (p(0,0) \wedge p(0,1)) \wedge (p(1,0) \wedge p(1,1)) \\ & \equiv (p(0,0) \wedge p(1,0)) \wedge (p(0,1) \wedge p(1,1)) \\ & \equiv \forall x\ p(x,0) \wedge \forall x\ p(x,1) \\ & \equiv \forall y \forall x\ p(x,y). \end{aligned}$$

We also have the following equivalences.

$$\begin{aligned} \exists x\ \exists y\ p(x,y) & \equiv \exists y\ p(0,y) \vee \exists y\ p(1,y) \\ & \equiv (p(0,0) \vee p(0,1)) \vee (p(1,0) \vee p(1,1)) \\ & \equiv (p(0,0) \vee p(1,0)) \vee (p(0,1) \vee p(1,1)) \\ & \equiv \exists x\ p(x,0) \vee \exists x\ p(x,1) \\ & \equiv \exists y\ \exists x\ p(x,y). \end{aligned}$$

We leave the proofs of equivalences (7.5) as exercises.

Equivalences containing Quantifiers and Connectives

It's time to start looking at some equivalences that involve quantifiers and connectives. For example, the following equivalence involves both quantifiers and the conditional connective. It shows that we can't always distribute a quantifier over a conditional.

An Equivalence to be Careful With **(7.6)**

$$\exists x\ (p(x) \rightarrow q(x)) \equiv \forall x\ p(x) \rightarrow \exists x\ q(x).$$

example

7.12 Proof of an Equivalence

We'll give a proof of (7.6) consisting of two subproofs showing that each side implies the other. First we'll prove the validity of

$$\exists x\ (p(x) \rightarrow q(x)) \rightarrow (\forall x\ p(x) \rightarrow \exists x\ q(x)).$$

Proof: Let I be a model for $\exists x\ (p(x) \rightarrow q(x))$ with domain D. Then $\exists x\ (p(x) \rightarrow q(x))$ is true for I, which means that $p(d) \rightarrow q(d)$ is true for some $d \in D$. Therefore, either $p(d)$ = false or $p(d) = q(d)$ = true for some $d \in D$. If $p(d)$ = false, then $\forall x\ p(x)$ is false for I; if $p(d) = q(d)$ = true, then $\exists x\ q(x)$ is true for I. In either case we obtain $\forall x\ p(x) \rightarrow \exists x\ q(x)$ is true for I. Therefore, I is a model for $\forall\ x\ p(x) \rightarrow \exists x\ q(x)$. QED.

Now we'll prove the validity of

$$(\forall x\ p(x) \rightarrow \exists x\ q(x)) \rightarrow \exists x\ (p(x) \rightarrow q(x)).$$

Proof: Let I be a model for $\forall x\ p(x) \rightarrow \exists x\ q(x)$ with domain D. Then $\forall x\ p(x) \rightarrow \exists x\ q(x)$ is true for I. Therefore, either $\forall x\ p(x)$ is false for I or both $\forall x\ p(x)$ and $\exists x\ q(x)$ are true for I. If $\forall x\ p(x)$ is false for I, then $p(d)$ is false for some $d \in D$. Therefore, $p(d) \rightarrow q(d)$ is true. If both $\forall x\ p(x)$ and $\exists x\ q(x)$ are true for I, then there is some $c \in D$ such that $p(c) = q(c)$ = true. Thus $p(c) \rightarrow q(c)$ is true. So in either case, $\exists x\ (p(x) \rightarrow q(x))$ is true for I. Thus I is a model for $\exists x(p(x) \rightarrow q(x))$. QED.

end example

Of course, once we know some equivalences, we can use them to prove other equivalences. For example, let's see how previous results can be used to prove the following equivalences.

Distributing the Quantifiers **(7.7)**

a. $\exists x\ (p(x) \vee q(x)) \equiv \exists x\ p(x) \vee \exists x\ q(x)$.

b. $\forall x\ (p(x) \wedge q(x)) \equiv \forall x\ p(x) \wedge \forall x\ q(x)$.

Proof of (7.7a):

$$\begin{aligned}\exists x\,(p(x) \vee q(x)) &\equiv \exists x\,(\neg\, p(x) \rightarrow q(x)) & \\ &\equiv \forall x\, \neg\, p(x) \rightarrow \exists x\, q(x) & (7.6)\\ &\equiv \neg\, \exists x\, p(x) \rightarrow \exists x\, q(x) & (7.4)\\ &\equiv \exists x\, p(x) \vee \exists x\, q(x) & \text{QED.}\end{aligned}$$

Proof of (7.7b): Use the fact that $\forall x\ (p(x) \wedge q(x)) \equiv \neg\, \exists x\ (\neg\, p(x) \vee \neg\, q(x))$ and then apply (7.7a). QED.

Restricted Equivalences

Some interesting and useful equivalences can occur when certain restrictions are placed on the variables. To start things off, we'll see how to change the name of a quantified variable in a wff without changing the meaning of the wff.

We'll illustrate the renaming problem with the following interpreted wff to represent the fact over the integers that for every integer x there is an integer y greater than x:

$$\forall x\ \exists y\ x < y.$$

Can we replace all ocurrences of the quantifier variable x with some other variable? If we choose a variable different from x and y, say z, we obtain

$$\forall z\ \exists y\ z < y.$$

This looks perfectly fine. But if we choose y to replace x, then we obtain

$$\forall y\ \exists y\ y < y.$$

This looks bad. Not only has $\forall y$ lost its influence, but the statement says there is an integer y such that $y < y$, which is false. So we have to be careful when renaming quantified variables. We'll always be on solid ground if we pick a new variable that does not occur anywhere in the wff. Here's the rule.

Renaming Rule (7.8)

If y is a new variable that does not occur in $W(x)$, then the following equivalences hold:

a. $\exists x\ W(x) \equiv \exists y\ W(x/y)$.

b. $\forall x\ W(x) \equiv \forall y\ W(x/y)$.

Remember that $W(x/y)$ is obtained from $W(x)$ by replacing all free occurrences of x by y.

example

7.13 Renaming Variables

We'll rename the quantified variables in the following wff so that they are all distinct:

$$\forall x\ \exists y\ (p(x, y) \to \exists x\ q(x, y) \vee \forall y\ r(x, y)).$$

Since there are four quantifiers using just the two variables x and y, we need two new variables, say z and w, which don't occur in the wff. We can replace any of the quantified wffs. So we'll start by replacing $\forall x$ by $\forall\ z$ and each x bound to $\forall x$ by z to obtain the following equivalent wff:

$$\forall z\ \exists y\ (p(z, y) \to \exists x\ q(x, y) \vee \forall y\ r(z, y)).$$

Notice that $p(x, y)$ and $r(x, y)$ have changed to $p(z, y)$ and $r(z, y)$ because the scope of $\forall\ x$ is the entire wff while the scope of $\exists x$ is just $q(x, y)$. Now let's replace $\forall y\ r(z, y)$ by $\forall w\ r(z, w)$ to obtain the following equivalent wff:

$$\forall z\ \exists y\ (p(z, y) \to \exists x\ q(x, y) \vee \forall w\ r(z, w)).$$

We end up with an equivalent wff with distinct quantified variables.

end example

Now we'll look at some restricted equivalences that allow us to move a quantifier past a wff that doesn't contain a free occurrence of the quantified variable.

Equivalences with Restrictions

If x does not occur free in C, *then* the following equivalences hold.

Simplification (7.9)

$\forall x\ C \equiv C$ and $\exists x\ C \equiv C$.

Disjunction (7.10)

a. $\forall x\ (C \vee A(x)) \equiv C \vee \forall x\ A(x)$.

b. $\exists x\ (C \vee A(x)) \equiv C \vee \exists x\ A(x)$.

Conjunction (7.11)

a. $\forall x\ (C \wedge A(x)) \equiv C \wedge \forall x\ A(x)$.

b. $\exists x\ (C \wedge A(x)) \equiv C \wedge \exists x\ A(x)$.

Continued ➡

➡ ➡

Implication (7.12)

a. $\forall x\ (C \to A(x)) \equiv C \to \forall x\ A(x).$

b. $\exists x\ (C \to A(x)) \equiv C \to \exists x\ A(x).$

c. $\forall x\ (A(x) \to C) \equiv \exists x\ A(x) \to C.$

d. $\exists x\ (A(x) \to C) \equiv \forall x\ A(x) \to C.$

Proof: We'll prove (7.10a). The important point in this proof is the assumption that x is not free in C. This means that any substitution x/t does not change C. In other words, $C(x/t) = C$ for all possible terms t. We'll assume that I is an interpretation with domain D. With these assumptions we can start.

If I is a model for $\forall x\ (C \vee A(x))$, then $(C \vee A(x))(x/d)$ is true with respect to I for all d in D. Now write $(C \vee A(x))(x/d)$ as

$$\begin{aligned}(C \vee A(x))(x/d) &= C(x/d) \vee A(x)(x/d) && \text{(substitution property)}\\ &= C \vee A(x)(x/d) && \text{(because } x \text{ is not free in } C).\end{aligned}$$

So $C \vee A(x)(x/d)$ is true for I for all d in D. Since the truth of C is not affected by any substitution for x, it follows that either C is true for I or $A(x)(x/d)$ is true for I for all d in D. So either I is a model for C or I is a model for $\forall x\ A(x)$. Therefore, I is a model for $C \vee \forall x\ A(x)$.

Conversely, if I is a model for $C \vee \forall x\ A(x)$, then $C \vee \forall x\ A(x)$ is true for I. Therefore, either C is true for I or $\forall x\ A(x)$ is true for I. Suppose that C is true for I. Then, since x is not free in C, we have $C = C(x/d)$ for any d in D. So $C(x/d)$ is true for I for all d in D. Therefore, $C(x/d) \vee A(x)(x/d)$ is also true for I for all d in D. Subtitution gives $C(x/d) \vee A(x)(x/d) = (C \vee A(x))(x/d)$. So $(C \vee A(x))(x/d)$ is true for I for all d in D. This means I is a model for $\forall x\ (C \vee A(x))$. Suppose $\forall x\ A(x)$ is true for I. Then $A(x)(x/d)$ is true for I for all d in D. So $C(x/d) \vee A(x)(x/d)$ is true for I for all d in D, and thus $(C \vee A(x))(x/d)$ is true for I for all d in D. So I is a model for $C \vee \forall x\ A(x)$. QED.

The proof of (7.10b) is similar and we'll leave it as an exercise. Once we have the equivalences (7.10), the other equivalences are simple consequences. For example, we'll prove (7.11b):

$$\begin{aligned}\exists x\,(C \wedge A(x)) &\equiv \neg\ \forall x\ \neg\ (C \wedge A(x)) && (7.4)\\ &\equiv \neg\ \forall x\,(\neg\ C \vee \neg\ A(x)) && \\ &\equiv \neg\ (\neg\ C \vee \forall x\ \neg\ A(x)) && (7.10a)\\ &\equiv \neg\ (\neg\ C \vee \neg\ \exists x\ A(x)) && (7.4)\\ &\equiv C \wedge \exists x\ A(x) &&\end{aligned}$$

The implication equivalences (7.12) are also easily derived from the other equivalences. For example, we'll prove (7.12c):

$$\begin{aligned}\forall x\,(A(x) \to C) &\equiv \forall x\,(\neg\, A(x) \vee C) & \\ &\equiv \forall x\, \neg\, A(x) \vee C & (7.10a)\\ &\equiv \neg\, \exists x\; A(x) \vee C & (7.4)\\ &\equiv \exists x\; A(x) \to C. &\end{aligned}$$

Now that we have some equivalences on hand, we can use them to prove other equivalences. In other words, we have a set of rules to transform wffs into other wffs having the same meaning. This justifies the word "calculus" in the name "predicate calculus."

7.2.2 Normal Forms

In the propositional calculus we know that any wff is equivalent to a wff in conjunctive normal form and to a wff in disjunctive normal form. Let's see whether we can do something similar with the wffs of the predicate calculus. We'll start with a definition. A wff W is in *prenex normal form* if all its quantifiers are on the left of the expression. In other words, a prenex normal form looks like the following:

$$Q_1 x_1 \ldots\ Q_n x_n\ M,$$

where each Q_i is either $\forall$ or $\exists$, each x_i is distinct, and M is a wff without quantifiers. For example, the following wffs are in prenex normal form:

$$\begin{aligned}&p(x),\\ &\exists x\; p(x),\\ &\forall x\; p(x,y),\\ &\forall x\; \exists y\,(p(x,y) \to q(x)),\\ &\forall x\; \exists y\; \forall z\,(p(x) \vee q(y) \wedge r(x,z)).\end{aligned}$$

Is any wff equivalent to some wff in prenex normal form? Yes. In fact there's an easy algorithm to obtain the desired form. The idea is to make sure that variables have distinct names and then apply equivalences that send all quantifiers to the left end of the wff. Here's the algorithm:

Prenex Normal Form Algorithm **(7.13)**

Any wff W has an equivalent prenex normal form, which can be constructed as follows:

1. Rename the variables of W so that no quantifiers use the same variable name and such that the quantified variable names are distinct from the free variable names.
2. Move quantifiers to the left by using equivalences (7.4), (7.10), (7.11), and (7.12).

The renaming of variables is important to the success of the algorithm. For example, we can't replace $p(x) \vee \forall x\ q(x)$ by $\forall x\ (p(x) \vee q(x))$ because they aren't equivalent. But we can rename variables to obtain the following equivalence:

$$p(x) \vee \forall x\ q(x) \equiv p(x) \vee \forall y\ q(y) \equiv \forall y\ (p(x) \vee q(y)).$$

example 7.14 Prenex Normal Form

We'll put the following wff W into prenex normal form:

$$A(x) \wedge \forall x\ (B(x) \rightarrow \exists y\ C(x,\ y) \vee \neg\ \exists y\ A(y)).$$

First notice that y is used in two quantifiers and x occurs both free and in a quantifier. After changing names, we obtain the following equivalent wff:

$$A(x) \wedge \forall z\ (B(z) \rightarrow \exists y\ C(z,\ y) \vee \neg\ \exists w\ A(w)).$$

Now each quantified variable is distinct, and the quantified variables are distinct from the free variable x. We'll apply equivalences to move all the quantifiers to the left:

$$\begin{aligned}
W &\equiv A(x) \wedge \forall z\,(B(z) \rightarrow \exists y\ C(z,y) \vee \neg\ \exists w\ A(w)) & \\
&\equiv \forall z\,(A(x) \wedge (B(z) \rightarrow \exists y\ C(z,y) \vee \neg\ \exists w\ A(w))) & (7.11)\\
&\equiv \forall z\,(A(x) \wedge (B(z) \rightarrow \exists y\,(C(z,y) \vee \neg\ \exists w\ A(w)))) & (7.10)\\
&\equiv \forall z\,(A(x) \wedge \exists y\,(B(z) \rightarrow C(z,y) \vee \neg\ \exists w\ A(w))) & (7.12)\\
&\equiv \forall z\ \exists y\,(A(x) \wedge (B(z) \rightarrow C(z,y) \vee \neg\ \exists w\ A(w))) & (7.11)\\
&\equiv \forall z\ \exists y\,(A(x) \wedge (B(z) \rightarrow C(z,y) \vee \forall w\ \neg\ A(w))) & (7.4)\\
&\equiv \forall z\ \exists y\,(A(x) \wedge (B(z) \rightarrow \forall w\,(C(z,y) \vee \neg\ A(w)))) & (7.10)\\
&\equiv \forall z\ \exists y\,(A(x) \wedge \forall w\,(B(z) \rightarrow C(z,y) \vee \neg\ A(w))) & (7.12)\\
&\equiv \forall z\ \exists y\ \forall w\,(A(x) \wedge (B(z) \rightarrow C(z,y) \vee \neg\ A(w))) & (7.11)
\end{aligned}$$

This wff is in the desired prenex normal form.

end example

There are two special prenex normal forms that correspond to the disjunctive normal form and the conjunctive normal form for propositional calculus. We define a *literal* in the predicate calculus to be an atom or the negation of an atom. For example, $p(x)$ and $\neg\ q(x,\ y)$ are literals. A prenex normal form is called a *prenex disjunctive normal form* if it has the form

$$Q_1x_1 \ldots\ Q_nx_n\ (C_1 \vee \cdots \vee C_k),$$

where each C_i is a conjunction of one or more literals. Similarly, a prenex normal form is called a *prenex conjunctive normal form* if it has the form

$$Q_1x_1 \ldots\ Q_nx_n\ (D_1 \wedge \cdots \wedge D_k),$$

where each D_i is a disjunction of one or more literals.

It's easy to construct either of these normal forms from a prenex normal form. Just eliminate conditionals, move $\neg$ inwards, and either distribute $\wedge$ over $\vee$ or distribute $\vee$ over $\wedge$. If we want to start with an arbitrary wff, then we can put everything together in a nice little algorithm. We can save some thinking by removing all conditionals at an early stage of the process. Then we won't have to remember the formulas (7.12). The algorithm can be stated as follows:

Prenex Disjunctive/Conjunctive Normal Form Algorithm (7.14)

Any wff W has an equivalent prenex disjunctive/conjunctive normal form, which can be constructed as follows:

1. Rename the variables of W so that no quantifiers use the same variable name and such that the quantified variable names are distinct from the free variable names.
2. Remove implications by using the equivalence $A \to B \equiv \neg\ A \vee B$.
3. Move negations to the right to form literals by using the equivalences (7.4) and the equivalences $\neg\ (A \wedge B) \equiv \neg\ A \vee \neg\ B$, $\neg\ (A \vee B) \equiv \neg\ A \wedge \neg\ B$, and $\neg\ \neg\ A \equiv A$.
4. Move quantifiers to the left by using equivalences (7.10) and (7.11).
5. To obtain the disjunctive normal form, distribute $\wedge$ over $\vee$. To obtain the conjunctive normal form, distribute $\vee$ over $\wedge$.

Now let's do an example that uses (7.14) to transform a wff into prenex normal form.

example 7.15 Prenex CNF and DNF

Let W be the following wff, which is the same wff from Example 7.14:

$$A(x) \wedge \forall x\ (B(x) \to \exists y\ C(x,\ y) \vee \neg\ \exists y\ A(y)).$$

We'll use algorithm (7.14) to construct a prenex conjunctive normal form and a prenex disjunctive normal form of W.

$$
\begin{aligned}
W &= A(x) \wedge \forall x\,(B(x) \rightarrow \exists y\; C(x,y) \vee \neg\, \exists y\; A(y)) & \\
&\equiv A(x) \wedge \forall z\,(B(z) \rightarrow \exists y\; C(z,y) \vee \neg\, \exists w\; A(w)) & \text{(rename variables)} \\
&\equiv A(x) \wedge \forall z\,(\neg\, B(z) \vee \exists y\; C(z,y) \vee \neg\, \exists w\; A(w)) & \text{(remove } \rightarrow\text{)} \\
&\equiv A(x) \wedge \forall z\,(\neg\, B(z) \vee \exists y\; C(z,y) \vee \forall w\; \neg\, A(w)) & (7.4) \\
&\equiv \forall z\,(A(x) \wedge (\neg\, B(z) \vee \exists y\; C(z,y) \vee \forall w\; \neg\, A(w))) & (7.11) \\
&\equiv \forall z\,(A(x) \wedge \exists y\,(\neg\, B(z) \vee \exists y\; C(z,y) \vee \forall w\; \neg\, A(w))) & (7.10) \\
&\equiv \forall z\; \exists y\,(A(x) \wedge (\neg\, B(z) \vee C(z,y) \vee \forall w\; \neg\, A(w))) & (7.11) \\
&\equiv \forall z\; \exists y\,(A(x) \wedge \forall w\,(\neg\, B(z) \vee C(z,y) \vee \neg\, A(w))) & (7.10) \\
&\equiv \forall z\; \exists y\; \forall w\,(A(x) \wedge (\neg\, B(z) \vee C(z,y) \vee \neg\, A(w))) & (7.11)
\end{aligned}
$$

This wff is in prenex conjunctive normal form. We'll distribute $\wedge$ over $\vee$ to obtain the following prenex disjunctive normal form:

$$\equiv \forall z\; \exists y\; \forall w\; ((A(x) \wedge \neg\, B(z)) \vee (A(x) \wedge C(z,\, y)) \vee (A(x) \wedge \neg\, A(w))).$$

end example

7.2.3 Formalizing English Sentences

Now that we have a few tools at hand, let's see whether we can find some heuristics for formalizing English sentences. We'll look at several sentences dealing with people and the characteristics of being a politician and being crooked. Let $p(x)$ denote the statement "x is a politician," and let $q(x)$ denote the statement "x is crooked." For each of the following sentences we've listed a formalization with quantifiers. Before you look at each formalization, try to find one of your own. It may be correct, even though it doesn't look like the listed answer.

"Some politician is crooked."	$\exists x\; (p(x) \wedge q(x))$.
"No politician is crooked."	$\forall x\; (p(x) \rightarrow \neg\, q(x))$.
"All politicians are crooked."	$\forall x\; (p(x) \rightarrow q(x))$.
"Not all politicians are crooked."	$\exists x\; (p(x) \wedge \neg\, q(x))$.
"Every politician is crooked."	$\forall x\; (p(x) \rightarrow q(x))$.
"There is an honest politician."	$\exists x\; (p(x) \wedge \neg\, q(x))$.
"No politician is honest."	$\forall x\; (p(x) \rightarrow q(x))$.
"All politicians are honest."	$\forall x\; (p(x) \rightarrow \neg\, q(x))$.

Can we notice anything interesting about the formalizations of these sentences? Yes, we can. Notice that each formalization satisfies one of the following two properties:

The universal quantifier $\forall x$ quantifies a conditional.

The existential quantifier $\exists x$ quantifies a conjunction.

To see why this happens, let's look at the statement "Some politician is crooked." We came up with the wff $\exists x\ (p(x) \wedge q(x))$. Someone might argue that the answer could also be the wff $\exists x\ (p(x) \rightarrow q(x))$. Notice that the second wff is true even if there are no politicians, while the first wff is false in this case, as it should be. Another way to see the difference is to look at equivalent wffs. From (7.6) we have the equivalence $\exists x\ (p(x) \rightarrow q(x)) \equiv \forall x\ p(x) \rightarrow \exists x\ q(x)$. Let's see how the wff $\forall x\ p(x) \rightarrow \exists x\ q(x)$ reads when applied to our example. It says, "If everyone is a politician, then someone is crooked." This doesn't seem to convey the same thing as our original sentence.

Another thing to notice is that people come up with different answers. For example, the second sentence, "No politician is crooked," might also be written as follows:

$$\neg\ \exists x\ (p(x) \wedge q(x)).$$

It's nice to know that this answer is OK too because it's equivalent to the listed answer, $\forall x\ (p(x) \rightarrow \neg\ q(x))$. We'll prove the equivalence of the two wffs by applying (7.4) as follows:

$$\begin{aligned} \neg\ \exists x\,(p\,(x) \wedge q\,(x)) &\equiv \forall x \neg\ (p\,(x) \wedge q\,(x)) \\ &\equiv \forall x\,(\neg\, p\,(x) \vee \neg\, q\,(x)) \\ &\equiv \forall x\,(p\,(x) \rightarrow \neg\, q\,(x))\,. \end{aligned}$$

Of course, not all sentences are easy to formalize. For example, suppose we want to formalize the following sentence:

It is not the case that not every widget has no defects.

Suppose we let $w(x)$ mean "x is a widget" and let $d(x)$ mean "x has a defect." We might look at the latter portion of the sentence, which says, "every widget has no defects." We can formalize this statement as $\forall x\ (w(x) \rightarrow \neg\ d(x))$. Now the beginning part of the sentence says, "It is not the case that not." This is a double negation. So the formalization of the entire sentence is

$$\neg\ \neg\ \forall x\ (w(x) \rightarrow \neg\ d(x)),$$

which of course is equivalent to $\forall x\ (w(x) \rightarrow \neg\ d(x))$.

Let's discuss the little words "is" and "are." Their usage can lead to quite different formalizations. For example, the three statements

"4 is 2 + 2," "x is a widget," and "widgets are defective"

have the three formalizations $4 = 2 + 2$, $w(x)$, and $\forall x\ (w(x) \rightarrow d(x))$. So we have to be careful when we try to formalize English sentences.

As a final example, which we won't discuss, consider the following sentence taken from Section 2, Article I, of the Constitution of the United States of America.

No person shall be a Representative who shall not have attained to the Age of twenty-five Years, and been seven Years a Citizen of the United States, and who shall not, when elected, be an Inhabitant of that State in which he shall be chosen.

7.2.4 Summary

Here, all in one place, are some equivalences and restricted equivalences.

Equivalences

1. $\neg \forall x\ W(x) \equiv \exists x \neg\ W(x)$. (7.4)
2. $\neg \exists x\ W(x) \equiv \forall x \neg\ W(x)$. (7.4)
3. $\forall x\ \forall\ y\ W(x, y) \equiv \forall y\ \forall x\ W(x, y)$. (7.5)
4. $\exists x\ \exists y\ W(x, y) \equiv \exists y\ \exists x\ W(x, y)$. (7.5)
5. $\exists x\ (A(x) \to B(x)) \equiv \forall x\ A(x) \to \exists x\ B(x)$. (7.6)
6. $\exists x\ (A(x) \vee B(x)) \equiv \exists x\ A(x) \vee \exists x\ B(x)$. (7.7)
7. $\forall x\ (A(x) \wedge B(x)) \equiv \forall x\ A(x) \wedge \forall x\ B(x)$. (7.7)

Restricted Equivalences

The following equivalences hold if x does not occur free in the wff C:

Simplification

$\forall x\ C \equiv C$ and $\exists x\ C \equiv C$. (7.9)

Disjunction (7.10)

$\forall x\ (C \vee \mathrm{A}(x)) \equiv C \vee \forall x\ A(x)$.
$\exists x\ (C \vee \mathrm{A}(x)) \equiv C \vee \exists x\ A(x)$.

Conjunction (7.11)

$\forall x\ (C \wedge A(x)) \equiv C \wedge \forall x\ A(x)$.
$\exists x\ (C \wedge A(x)) \equiv C \wedge \exists x\ A(x)$.

Implication (7.12)

$\forall x\ (C \to A(x)) \equiv C \to \forall x\ A(x)$.
$\exists x\ (C \to A(x)) \equiv C \to \exists x\ A(x)$.
$\forall x\ (A(x) \to C) \equiv \exists x\ A(x) \to C$.
$\exists x\ (A(x) \to C) \equiv \forall x\ A(x) \to C$.

Exercises

Proving Equivalences with Validity

1. Prove each of the following equivalences with validity arguments (i.e., use interpretations and models).

   a. $\forall x\ (A(x) \wedge B(x)) \equiv \forall x\ A(x) \wedge \forall x\ B(x)$.
   b. $\exists x\ (A(x) \vee B(x)) \equiv \exists x\ A(x) \vee \exists x\ B(x)$.
   c. $\exists x\ (A(x) \rightarrow B(x)) \equiv \forall x\ A(x) \rightarrow \exists x\ B(x)$.
   d. $\forall x\ \forall y\ W(x, y) \equiv \forall y\ \forall\ x\ W(x, y)$.
   e. $\exists x\ \exists y\ W(x, y) \equiv \exists y\ \exists x\ W(x, y)$.

2. Assume that x does not occur free in the wff C. With this assumption, prove each of the following statements with a validity argument. In other words, do not use equivalences.

   a. $\forall x\ C \equiv C$.
   b. $\exists x\ C \equiv C$.
   c. $\exists x\ (C \vee A(x)) \equiv C \vee \exists x\ A(x)$.

Proving Equivalences with Equivalence

3. Assume that x does not occur free in the wff C. With this assumption, prove each of the following statements with an equivalence proof that uses the equivalence listed in parentheses.

   a. $\forall x\ (C \rightarrow A(x)) \equiv C \rightarrow \forall x\ A(x)$. (use 7.10a)
   b. $\exists x\ (C \rightarrow A(x)) \equiv C \rightarrow \exists x\ A(x)$. (use 7.10b)
   c. $\exists x\ (A(x) \rightarrow C) \equiv \forall x\ A(x) \rightarrow C$. (use 7.10b)
   d. $\forall x\ (C \wedge A(x)) \equiv C \wedge \forall x\ A(x)$. (use 7.10b)

Prenex Normal Forms

4. Use equivalences to construct a prenex conjunctive normal form for each of the following wffs.

   a. $\forall x\ (p(x) \vee q(x)) \rightarrow \forall x\ p(x) \vee \forall x\ q(x)$.
   b. $\exists x\ p(x) \wedge \exists x\ q(x) \rightarrow \exists x\ (p(x) \wedge q(x))$.
   c. $\forall x\ \exists y\ p(x, y) \rightarrow \exists y\ \forall x\ p(x, y)$.
   d. $\forall x\ (p(x, f(x)) \rightarrow p(x, y))$.
   e. $\forall x\ \forall y\ (p(x, y) \rightarrow \exists z\ (p(x, z) \wedge p(y, z)))$.
   f. $\forall x\ \forall y\ \forall z\ (p(x, y) \wedge p(y, z) \rightarrow p(x, z)) \wedge \forall x\ \neg\ p(x, x)$
   $\rightarrow \forall x\ \forall y\ (p(x, y) \rightarrow \neg\ p(y, x))$.

5. Use equivalences to construct a prenex disjunctive normal form for each of the following wffs.

a. $\forall x\ (p(x) \vee q(x)) \rightarrow \forall x\ p(x) \vee \forall x\ q(x)$.
b. $\exists x\ p(x) \wedge \exists x\ q(x) \rightarrow \exists x\ (p(x) \wedge q(x))$.
c. $\forall x\ \exists y\ p(x, y) \rightarrow \exists y\ \forall x\ p(x, y)$.
d. $\forall x\ (p(x, f(x)) \rightarrow p(x, y))$.
e. $\forall x\ \forall y\ (p(x, y) \rightarrow \exists z\ (p(x, z) \wedge p(y, z)))$.
f. $\forall x\ \forall y\ \forall z\ (p(x, y) \wedge p(y, z) \rightarrow p(x, z)) \wedge \forall x \neg\ p(x, x)$
$\rightarrow \forall x\ \forall y\ (p(x, y) \rightarrow \neg\ p(y, x))$.

6. Recall that an equivalence $A \equiv B$ stands for the wff $(A \rightarrow B) \wedge (B \rightarrow A)$. Let C be a wff that does not contain the variable x.

a. Find a countermodel to show that the following statement is invalid: $(\forall x\ W(x) \equiv C) \equiv \forall\ x\ (W(x) \equiv C)$.
b. Find a prenex normal form for the statement $(\forall x\ W(x) \equiv C)$.

Formalizing English Sentences

7. Formalize each of the following English sentences, where the domain of discourse is the set of all people, where $C(x)$ means x is a committee member, $G(x)$ means x is a college graduate, $R(x)$ means x is rich, $S(x)$ means x is smart, $O(x)$ means x is old, and $F(x)$ means x is famous.

a. Every committee member is rich and famous.
b. Some committee members are old.
c. All college graduates are smart.
d. No college graduate is dumb.
e. Not all college graduates are smart.

8. Formalize each of the following statements, where $B(x)$ means x is a bird, $W(x)$ means x is a worm, and $E(x, y)$ means x eats y.

a. Every bird eats worms.
b. Some birds eat worms.
c. Only birds eat worms.
d. Not all birds eat worms.
e. Birds only eat worms.
f. No bird eats only worms.
g. Not only birds eat worms.

9. Formalize each argument as a wff, where $P(x)$ means x is a person, $S(x)$ means x can swim, and $F(x)$ means x is a fish.

a. All fish can swim. John can't swim. Therefore, John is not a fish.
b. Some people can't swim. All fish can swim. Therefore, there is some person who is not a fish.

10. Formalize each statement, where $P(x)$ means x is a person, $B(x)$ means x is a bully, $K(x, y)$ means x is kind to y, $C(x)$ means x is a child, $A(x)$ means x is an animal, $G(x)$ means x plays golf, and $N(x, y)$ means x knows y.
    a. All people except bullies are kind to children.
    b. Bullies are not kind to children.
    c. Bullies are not kind to themselves.
    d. Not everyone plays golf.
    e. Everyone knows someone who plays golf.
    f. People who play golf are kind to animals.
    g. People who are not kind to animals do not play golf.

7.3 Formal Proofs in Predicate Calculus

To reason formally about wffs in the predicate calculus, we need some inference rules. It's nice to know that all the inference rules of the propositional calculus can still be used for the predicate calculus. We just need to replace "tautology" with "valid." In other words, if R is an inference rule for the propositional calculus that maps tautologies to a tautology, then R also maps valid wffs to a valid wff.

For example, let's take the modus ponens inference rule of the propositional calculus and prove that it also works for the predicate calculus. In other words, we'll show that modus ponens maps valid wffs to a valid wff.

Proof: Let A and $A \to B$ be valid wffs. We need to show that B is valid. Suppose we have an interpretation for B with domain D. We can use D to give an interpretation to A by assigning values to all the predicates, functions, free variables, and constants that occur in A but not B. This gives us interpretations for A, B, and $A \to B$ over the domain D. Since we are assuming that A and $A \to B$ are valid, it follows that A and $A \to B$ are true for these interpretations over D. Now we can apply the modus ponens rule for propositions to conclude that B is true with respect to the given interpretation over D. Since the given interpretation of B was arbitrary, it follows that every interpretation of B is a model. Therefore, B is valid. QED.

We can use similar arguments to show that all inference rules of the propositional calculus are also inference rules of the predicate calculus. So we have a built-in collection of rules to do formal reasoning in the predicate calculus. But we need more.

Sometimes it's hard to reason about statements that contain quantifiers. The natural approach is to remove quantifiers from statements, do some reasoning with the unquantified statements, and then restore any needed quantifiers. We might call this the RRR method of reasoning with quantifiers—*remove*, *reason*, and *restore*. But quantifiers cannot be removed and restored at will. There are some restrictions that govern their use. So we'll spend a little time discussing them.

Although there are restrictions on the use of quantifiers, the nice thing is that if we use the rules properly, we can continue to use conditional proof in the predicate calculus. In other words, the conditional proof rule (CP) (i.e., the deduction theorem) for propositional calculus carries over to predicate calculus. We can also use the indirect proof rule (IP). Now let's get on to the four quantifier rules.

7.3.1 Universal Instantiation (UI)

Let's start by using our intuition and see how far we can get. It seems reasonable to say that if a property holds for everything, then it holds for any particular thing. In other words, we should be able to infer $W(x)$ from $\forall x\ W(x)$. Similarly, we should be able to infer $W(c)$ from $\forall x\ W(x)$ for any constant c.

Can we infer $W(t)$ from $\forall x\ W(x)$ for any term t? This seems OK too, but there may be a problem if $W(x)$ contains a free occurrence of x that lies within the scope of a quantifier. For example, suppose we let

$$W(x) = \exists y\ p(x, y).$$

Now if we let $t = y$, then we obtain

$$W(t) = W(y) = \exists y\ p(y, y).$$

But we can't always infer $\exists y\ p(y, y)$ from $\forall x\ \exists y\ p(x, y)$. For example, let $p(x, y)$ mean "x is a child of y." Then the statement $\forall x\ \exists y\ p(x, y)$ is true because every person x is a child of some person y. But the statement $\exists y\ p(y, y)$ is false because no person is their own child.

Trouble arises when we try to infer $W(t)$ from $\forall x\ W(x)$ in situations where t contains an occurrence of a quantified variable and x occurs free within the scope of that quantifier. We must restrict our inferences so that this does not happen. To make things precise, we'll make the following definition.

Definition of Free to Replace

We say that the term t *is free to replace* x in $W(x)$ if either no variable of t occurs bound to a quantifier in $W(x)$ or x does not occur free within the scope of a quantifier in $W(x)$. Equivalently we can say that a term t is free to replace x in $W(x)$ if both $W(t)$ and $W(x)$ have the same bound occurrences of variables.

example 7.16 Free to Replace

A term t is free to replace x in $W(x)$ under any of the following conditions.

a. $t = x$.

b. t is a constant.

c. The variables of t do not occur in $W(x)$.

d. The variables of t do not occur within the scope of a quantifier in $W(x)$.

e. x does not occur free within the scope of a quantifier in $W(x)$.

f. $W(x)$ does not have any quantifiers.

end example

Going the other way, we can say that a term t is *not* free to replace x in $W(x)$ if t contains a variable that is quantifed in $W(x)$ *and* x occurs free within the scope of that quantifier. We can also observe that when t is not free to replace x in $W(x)$, then $W(t)$ has more bound variables than $W(x)$, while if t is free to replace x in $W(x)$, then $W(t)$ and $W(x)$ have the same bound variables.

example 7.17 Not Free to Replace

We'll examine some terms t that are not free to replace x in $W(x)$, where $W(x)$ is the following wff:

$$W(x) = q(x) \wedge \exists y\, p(x, y).$$

We'll examine the following two terms:

$$t = y \text{ and } t = f(x, y).$$

Notice that both terms contain the variable y, which is quantified in $W(x)$ and there is a free occurrence of x within the scope of the quantifier $\exists y$. So each t is not free to replace x in $W(x)$. We can also note the difference in the number of bound variables between $W(t)$ and $W(x)$. For example, for $t = y$ we have

$$W(t) = W(y) = q(y) \wedge \exists y\, p(y, y).$$

So $W(t)$ has one more bound occurrence of y than $W(x)$. The same property holds for $t = f(x, y)$.

end example

Now we're in a position to state the universal instantiation rule along with the restrictions on its use.

Universal Instantiation Rule (UI) **(7.15)**

$$\frac{\forall x W(x)}{\therefore W(t)} \quad \textit{Restriction: } t \text{ is free to replace } x \text{ in } W(x).$$

Special cases where the restriction is always satisfied:

$$\frac{\forall x W(x)}{\therefore W(x)} \quad \text{and} \quad \frac{\forall x W(x)}{\therefore W(c)} \quad (\text{for any constant } c).$$

Proof: The key point in the proof comes from the observation that if t is free to replace x in $W(x)$, then for any interpretation, the interpretation of $W(t)$ is the same as the interpretation of $W(d)$, where d is the interpreted value of t. To state this more concisely for an interpretation I, let tI be the interpreted value of t by I, let $W(t)I$ be the interpretation of $W(t)$ by I, and let $W(tI)I$ be the interpretation of $W(tI)$ by I. Now we can state the key point as an equation. If t is free to replace x in $W(x)$, then for any interpretation I, the following equation holds:

$$W(t)I = W(tI)I.$$

This can be proved by induction on the number of quantifiers and the number of connectives that occur in $W(x)$ and we'll leave it as an exercise. The UI rule follows easily from this. Let I be an interpretation with domain D that is a model for $\forall x\ W(x)$. Then $W(x)I$ is true for all $x \in D$. Since $tI \in D$, it follows that $W(tI)I$ is true. But we have the equation $W(t)I = W(tI)I$, so it also follows that $W(t)I$ is true. So I is a model for $W(t)$. Therefore, the wff $\forall x\ W(x) \to W(t)$ is valid. QED.

example 7.18 Meaning Is Preserved

We'll give some examples to show that $W(t)I = W(tI)I$ holds when t is free to replace x. Let $W(x)$ be the following wff:

$$W(x) = \exists y\ p(x, y, z).$$

Then, for any term t, we have

$$W(t) = \exists y\ p(t, y, z).$$

Let I be an interpretation with domain $D = \{a, b, c\}$ that assigns any free occurrences of x, y, and z to a, b, and c. In each of the following examples, t is free to replace x in $W(x)$.

1. $t = b$: $W(t)I = \exists y\ p(b,y,z)I = \exists y\ p(b,y,c) = W(b)I = W(tI)I.$
2. $t = x$: $W(t)I = \exists y\ p(x,y,z)I = \exists y\ p(a,y,c) = \exists y\ p(a,y,z)I = W(a)I = W(tI)I.$
3. $t = z$: $W(t)I = \exists y\ p(z,y,z)I = \exists y\ p(c,y,c) = \exists y\ p(c,y,z)I = W(c)I = W(tI)I.$
4. $t = f(x,z)$: $W(t)I = \exists y\ p(f(x,z),y,z)I$
$= \exists y\ p(f(a,c),y,c)$
$= \exists y\ p(f(a,c),y,z)I$
$= W(f(a,c))I = W(tI)I.$

We should observe that the equations hold no matter what meaning we give to f and p.

end example

example 7.19 Meaning Is Not Preserved

We'll give some examples to show that $W(t)I \neq W(tI)I$ when t is not free to replace x in $W(x)$. We'll use same wff from Example 7.18.

$$W(x) = \exists y\ p(x, y, z).$$

Notice that each of the following terms t is not free to replace x in $W(x)$ because each one contains the quantified variable y and x occurs free in the scope of the quantifier.

$$t = y \quad \text{and} \quad t = f(x, y).$$

Let I be the interpretation with domain $D = \{1, 2, 3\}$ that assigns any free occurrences of x, y, and z to 1, 2, and 3, respectively. Let $p(u, v, w)$ mean that u, v, and w are all distinct and let $f(u, v)$ be the maximum of u and v.

1. $t = y$:

$$\begin{aligned} W(t)I &= W(y)I \\ &= \exists y\ p(y, y, z)I \\ &= \exists y\ p(y, y, 3), \text{ which is false.} \end{aligned}$$

$$\begin{aligned} W(tI)I &= W(yI)I \\ &= W(2)I \\ &= \exists y\ p(2, y, z)I \\ &= \exists y\ p(2, y, 3), \text{ which is true.} \end{aligned}$$

2. $t = f(x, y)$:

$$\begin{aligned} W(t)I &= W(f(x, y))I \\ &= \exists y\ p(f(x, y), y, z)I \\ &= \exists y\ p(f(1, y), y, 3), \text{ which is false (try different y's).} \end{aligned}$$

$$\begin{aligned} W(tI)I &= W(f(x, y)I)I \\ &= W(f(1, 2)I)I \\ &= W(2)I \\ &= \exists y\ p(2, y, z)I \\ &= \exists y\ p(2, y, 3), \text{ which is true.} \end{aligned}$$

So in each case $W(t)I$ and $W(tI)I$ have different truth values.

end example

7.3.2 Existential Generalization (EG)

It seems to make sense that if a property holds for a particular thing, then the property holds for some thing. For example, we know that 5 is a prime number, so it makes sense to conclude that there is some prime number. In other words, if we let $p(x)$ mean "x is a prime number," then from $p(5)$ we can infer $\exists x\ p(x)$. So far, so good. If a wff can be written in the form $W(t)$ for some term t, can we infer $\exists x\ W(x)$?

After a little thought, this appears to be related to the UI rule in its contrapositive form. In other words, notice the following equivalences:

$$\begin{aligned} W(t) \to \exists x\ W(x) &\equiv \neg\ \exists x\ W(x) \to \neg\ W(t) \\ &\equiv \forall x \neg\ W(x) \to \neg\ W(t)\,. \end{aligned}$$

The last wff is an instance of the UI rule, which tells us that the wff is valid if t is free to replace x in $\neg\ W(x)$. Since $W(x)$ and $\neg\ W(x)$ differ only by negation, it follows that t is free to replace x in $\neg\ W(x)$ if and only if t is free to replace x in $W(x)$. Therefore, we can say that

$W(t) \to \exists x\ W(x)$ is valid if t is free to replace x in $W(x)$.

So we have the appropriate restriction for removing the existential quantifier.

Existential Generalization Rule **(7.16)**

$$\frac{W(t)}{\therefore \exists\, x\ W(x)}$$ *Restriction:* t is free to replace x in $W(x)$.

Special cases where the restriction is always satisfied:

$$\frac{W(x)}{\therefore \exists\, x\ W(x)} \quad \text{and} \quad \frac{W(c)}{\therefore \exists\, x\ W(x)} \quad \text{(for any constant } c\text{).}$$

Usage Note

There is a kind of forward-backward reasoning to keep in mind when using the EG rule. If we want to apply EG to a wff, then we must be able to write the wff in the form $W(t)$ for some term t. This means, as always, that we must be able to write the wff in the form $W(x)$ for some variable x such that $W(t)$ is obtained from $W(x)$ by replacing all free occurrences of x by t. Once this is done, we check to see whether t is free to replace x in $W(x)$.

example **7.20 Using the EG Rule**

Let's examine the use of EG on some sample wffs. We'll put each wff into the form $W(t)$ for some term t, where $W(t)$ is obtained from a wff $W(x)$ for some variable x, and t is free to replace x in $W(x)$. Then we'll use EG to infer $\exists x\ W(x)$.

1. $\forall y\ p(c, y)$.

We can write $\forall y\ p(c, y) = W(c)$, where $W(x) = \forall y\ p(x, y)$. Now since c is a constant, we can use EG to infer

$$\exists x\ \forall y\ p(x, y).$$

For example, over the domain of natural numbers, let $p(x, y)$ mean $x \leq y$ and let $c = 0$. Then from $\forall y\ (0 \leq y)$ we can use EG to infer $\exists x\ \forall y\ (x \leq y)$.

2. $p(x, y, c)$.

We can write $p(x, y, c) = W(c)$, where $W(z) = p(x, y, z)$. Since c is a constant, the EG rule can be used to infer

$$\exists z\ p(x, y, z).$$

Notice that we can also write $p(x, y, c)$ as either $W(x)$ or $W(y)$ with no substitutions. So EG can also be used to infer $\exists x\ p(x, y, c)$ and $\exists y\ p(x, y, c)$.

3. $\forall y\ p(f(x, z), y)$.

We can write $\forall y\ p(f(x, z), y) = W(f(x, z))$, where $W(x) = \forall y\ p(x, y)$. Notice that the term $f(x, z)$ is free to replace x in $W(x)$. So we can use EG to infer

$$\exists x\ \forall y\ p(x, y).$$

We can also write $\forall y\ p(f(x, z), y)$ as either of the forms $W(x)$ or $W(z)$ with no substitutions. Therefore, we can use EG to infer $\exists x\ \forall\ y\ p(f(x, z), y)$ and $\exists z\ \forall y\ p(f(x, z), y)$.

end example

7.3.3 Existential Instantiation (EI)

It seems reasonable to say that if a property holds for some thing, then it holds for a particular thing. This type of reasoning is used quite often in proofs that proceed in the following way. Assume that we are proving some statement and during the proof we have the wff

$$\exists x\ W(x).$$

We then say that $W(c)$ holds for a particular object c. From this point the proof proceeds with more deductions and finally reaches a conclusion that does not contain any occurrence of the object c.

Although this may seem OK, we have to be careful about our choice of the constant. For example, if the wff $\exists x\, p(x, b)$ occurs in a proof, then we can't say that $p(b, b)$ holds. To see this, suppose we let $p(x, b)$ mean "x is a parent of b." Then $\exists x\, p(x, b)$ is true, but $p(b, b)$ is false because b is not a parent of b. So we can't pick a constant that is already in the wff.

But we need to restrict the choice of constant further. Suppose, for example, that we have the following partial "attempted" proof.

1.	$\exists x\, p(x)$	P
2.	$\exists x\, q(x)$	P
3.	$p(c)$	2, proposed EI rule
4.	$q(c)$	3, proposed EI rule
5.	$p(c) \wedge q(c)$	3, 4, Conj.
$\vdots$	$\vdots$	

This can't continue because line 5 does not follow from the premises. For example, suppose over the domain of integers we let $p(x)$ mean "x is odd" and let $q(x)$ mean "x is even." Then the premises are true because there is an odd number and there is an even number. But line 5 says that c is even and c is odd, which is a false statement. So we must pick a new constant distinct from any other constant in previous lines of the proof.

There is one more restriction on the constant that we introduce. Namely, the constant cannot appear in any conclusion. For example, suppose starting with the premise $\exists x\, p(x)$ we deduce $p(c)$ and then claim by conditional proof that we have proven the validity of the wff $\exists x\, p(x) \to p(c)$. But this wff is invalid. For example, consider the interpretation with domain $\{0, 1\}$, where we assign the constant $c = 1$ and let $p(0) =$ true and $p(1) =$ false. Then the interpreted wff has a true antecedent and a false consequent, which makes the interpreted wff false. Now we're in position to state the rule with the restrictions on its use.

Existential Instantiation Rule (7.17)

If c is a new constant in the proof and c does not occur in the conclusion of the proof, then

$$\frac{\exists\, x\, W(x)}{\therefore W(c)}.$$

Proof: We'll give an idea of the proof. Suppose that the EI rule is used in a proof of the wff A and the constant c does not occur in A. We'll show that the proof of A does not need the EI rule. (So there is no harm in using EI.) Let P be the conjunction of the premises in the proof, except $\exists x\ W(x)$ if it happens to be to be a premise. Therefore, the wff $P \wedge \exists x\ W(x) \wedge W(c) \rightarrow A$ is valid. So it follows that the wff $P \wedge \exists x\ W(x) \rightarrow (W(c) \rightarrow A)$ is also valid. Let y be a variable that does not occur in the proof. Then $P \wedge \exists x\ W(x) \rightarrow (W(y) \rightarrow A)$ is also valid because any interpretation assigning y a value yields the same wff by assigning c that value. It follows from the proof of (7.3) that $\forall y\ (P \wedge \exists x\ W(x) \rightarrow (W(y) \rightarrow A))$ is valid. Since y does not occur in $P \wedge \exists x\ W(x)$ or A, we have the following equivalences.

$$\begin{aligned}\forall y\,(P \wedge \exists x W(x) \rightarrow (W(y) \rightarrow A)) &\equiv P \wedge \exists x\ W(x) \rightarrow \forall y\,(W(y) \rightarrow A) && (7.12)\\ &\equiv P \wedge \exists x\ W(x) \rightarrow (\exists y\ W(y) \rightarrow A) && (7.12)\\ &\equiv P \wedge \exists x\ W(x) \wedge \exists y\ W(y) \rightarrow A\\ &\equiv P \wedge \exists x\ W(x) \rightarrow A.\end{aligned}$$

So A can be proved without the use of EI. QED.

7.3.4 Universal Generalization (UG)

It seems reasonable to say that if some property holds for an arbitary thing, then the property holds for all things. This type of reasoning is used quite often in proofs that proceed in the following way. We prove that some property holds for an arbitrary element x and then conclude that the property holds for all x. Here's a more detailed description of the technique in terms of a wff $W(x)$ over some domain D.

> We let x be an arbitrary but fixed element of the domain D. Next, we construct a proof that $W(x)$ is true. Then we say that since x was arbitrary, it follows that $W(x)$ is true for all x in D. So from a proof of $W(x)$, we have proved $\forall x\ W(x)$.

So we want to consider the possibility of generalizing a wff by attaching a universal quantifier. In other words, we want to consider the circumstances under which we can infer $\forall x\ W(x)$ from $W(x)$. If $W(x)$ is valid, then we can always do it because the two wffs are equivalent. For example, we have

$$p(x) \vee \neg\ p(x) \equiv \forall x\ (p(x) \vee \neg\ p(x)).$$

But the concern here is for situtations where $W(x)$ has been inferred as part of a proof. In this case we need to consider some restrictions.

Difficulty (Free Variables in a Premise)

Over the domain of natural numbers let $p(x)$ mean that x is a prime number. Then $\forall x\ p(x)$ means that every natural number x is a prime number. Since there are natural numbers that are not prime, we can't infer $\forall x\ p(x)$ from $p(x)$.

This illustrates a problem that can occur in a proof when there is a premise containing a free variable x and we later try to generalize with respect to the variable.

1.	$p(x)$	P
2.	$\forall x\, p(x)$	1, *Do not use the UG rule.* It doesn't work!

Restriction: Among the wffs used to infer $W(x)$, x is not free in any premise.

Difficulty (Free Variables and EI)

Another problem can occur when we try to generalize with respect to a variable that occurs free in a wff constructed with EI. For example, consider the following attempted proof, which starts with the premise that for any natural number x there is some natural number y greater than x.

1.	$\forall x\, \exists y\, (x < y)$	P
2.	$\exists y\, (x < y)$	1, UI
3.	$x < c$	2, EI
4.	$\forall x\, (x < c)$	3, *Do not use the UG rule.* It doesn't work.
5.	$\exists y\, \forall x\, (x < y)$	3, EG.
	Not QED	

We better not allow line 4 because the conclusion on line 5 says that there is a natural number y greater than every natural number x, which we know to be false.

Restriction: Among wffs used to infer $W(x)$, x is not free in any wff inferred by EI.

Now we're finally in position to state the *universal generalization rule* with its two restrictions:

Universal Generalization Rule (UG) **(7.18)**

$$\frac{W(x)}{\therefore \forall x\; W(x)}$$

Restrictions: Among the wffs used to infer $W(x)$, x is not free in any premise and x is not free in any wff constructed by EI.

Proof: We'll give a general outline that works if the restrictions are satisfied. Suppose we have a proof of $W(x)$. If we let P be the conjunction of premises in the proof, then $P \to W(x)$ is valid. We claim that $P \to \forall x\; W(x)$ is valid. For if not, then $P \to \forall x\; W(x)$ has some interpretation I with domain D such that P is true with respect to I and $\forall x\; W(x)$ is false with respect to I. So there is an element $d \in D$ such that $W(d)$ is false with respect to I. Now let J be the interpretation of $P \to W(x)$ with the same domain D and with all assignments the same as I but with the additional assignment of the free variable x to d. Since x is not free in P, it follows that P with respect to J is the same as P with respect to I. So P is true with respect to J. But since $P \to W(x)$ is valid, it follows that $W(d)$ is true with respect to J. But x is not free in $W(d)$. So $W(d)$ with respect to J is the same as $W(d)$ with respect to I, which contradicts $W(d)$ being false with respect to I. Therefore, $P \to \forall x\ W(x)$ is valid. So from $W(x)$ we can infer $\forall x\ W(x)$. QED.

It's nice to know that the restrictions of the UG rule are almost always satisfied. For example, if the premises in the proof don't contain any free variables and if the proof doesn't use the EI rule, then use the UG rule with abandon.

Conditional Proof Rule

Now that we've discussed the quantifier proof rules, let's take a minute to discuss the extension of the conditional proof rule from propositional calculus to predicate calculus. Recall that it allows us to prove a conditional $A \to B$ with a conditional proof of B from the premise A. The result is called the *conditional proof rule* (CP). It is also known as the deduction theorem.

Conditional Proof Rule (CP) **(7.19)**

If A is a premise in a proof of B, then there is a proof if $A \to B$ that does not use A as a premise.

Proof: Let $W_1, \ldots, W_n = B$ be a proof of B that contains A as a premise. We'll show by induction that for each k in the interval $1 \leq k \leq n$, there is a proof of $A \to W_k$ that does not use A as a premise. Since $B = W_n$, the result will be proved. For the case $k = 1$, the argument is the same as that given in the proof of the CP rule for propositional calculus. Let $k > 1$ and assume that for each $i < k$ there is a proof of $A \to W_i$ that does not use A as a premise. If W_k is not inferred by a quantifier rule, then the argument in the proof of the CP rule for propositional calculus constructs a proof of $A \to W_k$ that does not use A as a premise. The construction also guarantees that the premises needed in the proof of $A \to W_k$ are the premises other than A that are needed in the original proof of W_k.

Suppose that W_k is inferred by a quantifier rule from W_i, where $i < k$. First, notice that if A is not needed to prove W_i, then A is not needed to prove W_k. So we can remove A from the given proof of W_k. Now add the valid wff $W_k \to (A \to W_k)$ to the proof and then use MP to infer $A \to W_k$. This gives us a proof of $A \to W_k$ that does not use A as a premise. Second, notice from the proof of the EI rule (7.17) that for any proof that uses EI, there is an alternative proof that does not use EI. So we can assume that A is needed in the proof of W_i and EI is not used in the given proof.

If W_i infers W_k by UG, then $W_k = \forall x W_i$ where x is not free in any premise needed to prove W_i. So x is not free in A. Induction gives a proof of $A \to W_i$

that does not use A as a premise and x is not free in any premise needed to prove $A \to W_i$. So we can use UG to infer $\forall x\,(A \to W_i)$. Since x is not free in A, it follows from (7.12a) that $\forall x\,(A \to W_i) \to (A \to \forall x W_i)$ is valid. Now use MP to infer $A \to \forall x W_i$. So we have a proof of $A \to W_k$ that does not use A as premise.

If W_i infers W_k by UI, then there is a wff $C(x)$ such that $W_i = \forall x C(x)$ and $W_k = C(t)$, where t is free to replace x in $C(x)$. The proof of the UI rule tells us that $\forall x C(x) \to C(t)$ is valid. Induction gives a proof of $A \to \forall x C(x)$ that does not use A as a premise. Now use HS to infer $A \to C(t)$. So we have a proof of $A \to W_k$ that does not use A as premise.

If W_i infers W_k by EG, then there is a wff $C(x)$ such that $W_k = C(t)$ and $W_k = \exists x C(x)$, where t is free to replace x in $C(x)$. The proof of the EG rule tells us that $C(t) \to \exists x C(x)$ is valid. By induction there is a proof of $A \to C(t)$ that does not use A as a premise. Now use HS to infer $A \to \exists x C(x)$. So we have a proof of $A \to W_k$ that does not use A as premise. QED.

7.3.5 Examples of Formal Proofs

Finally we can get down to business and do some proofs. The following examples show the usefulness of the four quantifier rules. Notice in most cases that we can use the less restrictive forms of the rules.

example 7.21 Part of an Equivalence

We'll give an indirect formal proof of the following statement:

$$\forall x \neg W(x) \to \neg \exists x\, W(x).$$

Proof:			
	1.	$\forall x \neg W(x)$	P
	2.	$\neg \neg \exists x\, W(x)$	P for IP
	3.	$\exists x\, W(x)$	2, T
	4.	$W(c)$	3, EI
	5.	$\neg W(c)$	1, UI
	6.	$W(c) \wedge \neg W(c)$	4, 5, Conj
	7.	false	6, T
		QED	1, 2, 7, IP.

We'll prove the converse of $\forall x \neg W(x) \to \neg \exists x\, W(x)$ in Example 7.30.

end example

example 7.22 Using Hypothetical Syllogism

We'll prove the following statement:

$$\forall x\ (A(x) \to B(x)) \wedge \forall x\ (B(x) \to C(x)) \to \forall x\ (A(x) \to C(x)).$$

Proof:	1.	$\forall x\,(A(x) \rightarrow B(x))$	P
	2.	$\forall x\,(B(x) \rightarrow C(x))$	P
	3.	$A(x) \rightarrow B(x)$	1, UI
	4.	$B(x) \rightarrow C(x)$	2, UI
	5.	$A(x) \rightarrow C(x)$	3, 4, HS
	6.	$\forall x\,(A(x) \rightarrow C(x))$	5, UG
		QED	1, 2, 6, CP.

end example

example 7.23 Lewis Carroll's Logic

The following argument is from *Symbolic Logic* by Lewis Carroll.

> *Babies are illogical. Nobody is despised who can manage a crocodile. Illogical persons are despised. Therefore babies cannot manage crocodiles.*

We'll formalize the argument over the domain of people. Let $B(x)$ mean "x is a baby," $L(x)$ mean "x is logical," $D(x)$ mean "x is despised," and $C(x)$ mean "x can manage a crocodile." Then the four sentences become

$$\begin{aligned}
&\forall x\,(B(x) \rightarrow \neg\, L(x)).\\
&\forall x\,(C(x) \rightarrow \neg\, D(x)).\\
&\forall x\,(\neg\, L(x) \rightarrow D(x)).\\
&\text{Therefore, } \forall x\,(B(x) \rightarrow \neg\, C(x)).
\end{aligned}$$

Here is a formal proof that the argument is correct.

Proof:	1.	$\forall x\,(B(x) \rightarrow \neg\, L(x))$	P
	2.	$\forall x\,(C(x) \rightarrow \neg\, D(x))$	P
	3.	$\forall x\,(\neg\, L(x) \rightarrow D(x))$	P
	4.	$B(x) \rightarrow \neg\, L(x)$	1, UI
	5.	$C(x) \rightarrow \neg\, D(x)$	2, UI
	6.	$\neg\, L(x) \rightarrow D(x)$	3, UI
	7.	$B(x)$	P
	8.	$\neg\, L(x)$	4, 7, MP
	9.	$D(x)$	6, 8, MP
	10.	$\neg\, C(x)$	5, 9, MT
	11.	$B(x) \rightarrow \neg\, C(x)$	7, 10, CP
	12.	$\forall x\,(B(x) \rightarrow \neg\, C(x))$	11, UG
		QED	1, 2, 3, 12, CP.

Note that this argument holds for any interpretation. In other words, we've shown that the wff $A \rightarrow B$ is valid, where A and B are defined as follows:

$$\begin{aligned} A &= \forall x\,(B\,(x) \rightarrow \neg\, L\,(x)) \wedge \forall x\,(C\,(x) \rightarrow \neg\, D\,(x)) \wedge \forall x\,(\neg\, L\,(x) \rightarrow D\,(x))\,, \\ B &= \forall x\,(B\,(x) \rightarrow \neg\, C\,(x))\,. \end{aligned}$$

end example

example **7.24 Swapping Universal Quantifiers**

We'll prove the following general statement about swapping universal quantifiers: $\forall x\ \forall y\ W \rightarrow \forall y\ \forall x\ W$.

Proof:	1.	$\forall x\,\forall y\ W$	P
	2.	$\forall y\ W$	1, UI
	3.	W	2 , UI
	4.	$\forall x\ W$	3, UG
	5.	$\forall y\,\forall x\ W$	4, UG
		QED	1, 5, CP.

The converse of the statement can be proved in the same way. Therefore, we have a formal proof of the following equivalence in (7.5).

$$\forall x\ \forall y\ W \equiv \forall y\ \forall\ x\ W.$$

end example

example **7.25 Renaming Variables**

We'll give formal proofs of the equivalences that rename variables (7.8): Let $W(x)$ be a wff, and let y be a variable that does not occur in $W(x)$. Then the following renaming equivalences hold:

$$\begin{aligned} \exists x\ W(x) &\equiv \exists y\ W(y)\,, \\ \forall x\ W(x) &\equiv \forall y\ W(y)\,. \end{aligned}$$

First we'll prove $\exists x\ W(x) \equiv \exists y\ W(y)$, which will require proofs of

$$\exists x\ W(x) \rightarrow \exists y\ W(y) \quad \text{and} \quad \exists y\ W(y) \rightarrow \exists x\ W(x)\,.$$

Proof of $\exists x\ W(x) \to \exists y\ W(y)$:

1.	$\exists x\ W(x)$	P
2.	$W(c)$	1, EI
3.	$\exists y\ W(y)$	2, EG
	QED	1, 3, CP.

Proof of $\exists y\ W(y) \to \exists x\ W(x)$:

1.	$\exists y\ W(y)$	P
2.	$W(c)$	1, EI
3.	$\exists x\ W(x)$	2, EG
	QED	1, 3, CP.

Next, we'll prove the equivalence $\forall x\ W(x) \equiv \forall y\ W(y)$ by proving the two statements $\forall x\ W(x) \to \forall y\ W(y)$ and $\forall y\ W(y) \to \forall x\ W(x)$. We'll combine the two proofs into one proof as follows:

1.	$\forall x\ W(x)$	P	Start first proof.
2.	$W(y)$	1, UI	y is free to replace x.
3.	$\forall y\ W(y)$	2, UG	
4.	$\forall x\ W(x) \to \forall y\ W(y)$	1, 3, CP	Finish first proof.
5.	$\forall y\ W(y)$	P	Start second proof.
6.	$W(x)$	5, UI	x is free to replace y.
7.	$\forall x\ W(x)$	6, UG	
8.	$\forall y\ W(y) \to \forall x\ W(x)$	5, 7, CP	Finish second proof.
	QED	4, 8, T.	

end example

example

7.26 Using EI, UI, and EG

We'll prove the statement

$$\forall x\ p(x) \wedge \exists x\ q(x) \to \exists x\ (p(x) \wedge q(x)).$$

Proof:

1.	$\forall x\ p(x)$	P
2.	$\exists x\ q(x)$	P
3.	$q(c)$	2, EI
4.	$p(c)$	1, UI
5.	$p(c) \wedge q(c)$	3, 4, Conj
6.	$\exists x\ (p(x) \wedge q(x))$	5, EG
	QED	1, 2, 6, CP.

end example

example 7.27 Formalizing an Argument

Consider the following three statements:

Every computer science major is a logical thinker.

John is a computer science major.

Therefore, there is some logical thinker.

We'll formalize these statements as follows: Let $C(x)$ mean "x is a computer science major," let $L(x)$ mean "x is a logical thinker," and let the constant b mean "John." Then the three statements can be written more concisely as follows, over the domain of people:

$$\begin{array}{l} \forall x\,(C\,(x) \to L\,(x)) \\ C\,(b) \\ \therefore \exists x\ L\,(x)\,. \end{array}$$

These statements can be written as the following conditional wff:

$$\forall x\ (C(x) \to L(x)) \wedge C(b) \to \exists x\ L(x).$$

Although we started with a specific set of English sentences, we now have a wff of the first-order predicate calculus. We'll prove that this conditional wff is valid as follows:

Proof:			
	1.	$\forall x\ (C(x) \to L(x)) \wedge C(b)$	P
	2.	$\forall x\ (C(x) \to L(x))$	1, Simp
	3.	$C(b)$	1, Simp
	4.	$C(b) \to L(b)$	2, UI
	5.	$L(b)$	3, 4, MP
	6.	$\exists x\ L(x)$	5, EG
		QED	1, 6, CP.

end example

example 7.28 Formalizing an Argument

Let's consider the following argument:

All computer science majors are people.

Some computer science majors are logical thinkers.

Therefore, some people are logical thinkers.

We'll give a formalization of this argument. Let $C(x)$ mean "x is a computer science major," $P(x)$ mean "x is a person," and $L(x)$ mean "x is a logical thinker." Now the statements can be represented by the following wff:

$$\forall x\ (C(x) \to P(x)) \wedge \exists\ x\ (C(x) \wedge L(x)) \to \exists\ x\ (P(x) \wedge L(x)).$$

We'll prove that this wff is valid as follows:

Proof:			
	1.	$\forall x\ (C(x) \to P(x))$	P
	2.	$\exists x\ (C(x) \wedge L(x))$	P
	3.	$C(c) \wedge L(c)$	2, EI
	4.	$C(c) \to P(c)$	1, UI
	5.	$C(c)$	3, Simp
	6.	$P(c)$	4, 5, MP
	7.	$L(c)$	3, Simp
	8.	$P(c) \wedge L(c)$	6, 7, Conj
	9.	$\exists x\ (P(x) \wedge L(x))$	8, EG
		QED	1, 2, 9, CP.

end example

example **7.29 Move Quantifiers with Care**

We'll give a correct proof of the validity of the following wff:

$$\forall x\ A(x) \vee \forall x\ B(x) \to \forall x\ (A(x) \vee B(x)).$$

Proof:			
	1.	$\forall x\ A(x) \vee \forall x\ B(x)$	P
	2.	$\quad \forall x\ A(x)$	P
	3.	$\quad A(x)$	2, UI
	4.	$\quad A(x) \vee B(x)$	3, Add
	5.	$\quad \forall x\ (A(x) \vee B(x))$	4, UG
	6.	$\forall x A(x) \to \forall x\ (A(x) \vee B(x))$	2, 5, CP
	7.	$\quad \forall x\ B(x)$	P
	8.	$\quad B(x)$	7, UI
	9.	$\quad A(x) \vee B(x)$	8, Add
	10.	$\quad \forall x\ (A(x) \vee B(x))$	9, UG
	11.	$\forall x\ B(x) \to \forall x\ (A(x) \vee B(x))$	7, 10, CP
	12.	$\forall x\ (A(x) \vee B(x))$	1, 6, 11, CD
		QED	1, 12, CP.

end example

example **7.30 An Equivalence**

In Example 7.21 we gave a formal proof of the statement

$$\forall x \neg\ W(x) \to \neg\ \exists x\ W(x).$$

Now we're in a position to give a formal proof of its converse. Thus we'll have a formal proof of the following equivalence (7.4):

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

The converse that we want to prove is the wff $\neg \exists x\ W(x) \to \forall x \neg W(x)$. To prove this statement, we'll divide the proof into two parts. First, we'll prove the statement $\neg \exists x\ W(x) \to \neg W(x)$. Our proof will be indirect.

Proof:			
	1.	$\neg \exists x\ W(x)$	P
	2.	$W(x)$	P for IP
	3.	$\exists x\ W(x)$	2, EG
	4.	$\neg \exists x\ W(x) \wedge \exists x\ W(x)$	1, 3, Conj
	5.	false	4, T
		QED	1, 2, 5, IP.

Now we can easily prove the statement $\neg \exists x\ W(x) \to \forall\ x \neg W(x)$.

Proof:			
	1.	$\neg \exists x\ W(x)$	P
	2.	$\neg \exists x\ W(x) \to \neg W(x)$	T, proved above
	3.	$\neg W(x)$	1, 2, MP
	4.	$\forall x \neg W(x)$	3, UG
		QED	1, 4, CP.

end example

example 7.31 An Incorrect Proof

Suppose we're given the following wwf.

$$\exists x\ P(x) \wedge \exists x\ Q(x) \to \exists x\ (P(x) \wedge Q(x)).$$

This wff is not valid! For example, let $D = \{0, 1\}$, and set $P(0) = Q(1) =$ true and $P(1) = Q(0) =$ false. Then $\exists x\ P(x) \wedge \exists x\ Q(x)$ is true but $\exists x\ (P(x) \wedge Q(x))$ is false. We'll give an incorrect proof sequence that claims to show that the wff is valid.

1.	$\exists x\ P(x) \wedge \exists x\ Q(x)$	P
2.	$\exists x\ P(x)$	1, Simp
3.	$P(c)$	2, EI
4.	$\exists x\ Q(x)$	1, Simp
5.	$Q(c)$	4, EI *No:* c already occurs in line 3.
6.	$P(c) \wedge Q(c)$	3, 5, Conj
7.	$\exists x\ (P(x) \wedge Q(x))$	6, EG
	Not QED	1, 7, CP.

end example

example **7.32 Formalizing a Numerical Argument**

We'll formalize the following informal proof that the sum of any two odd integers is even. Proof: Let x and y be arbitrary odd integers. Then there exist integers m and n such that $x = 2m + 1$ and $y = 2n + 1$. Now add x and y to obtain

$$x + y = 2m + 1 + 2n + 1 = 2(m + n + 1)$$

Therefore, $x + y$ is an even integer. Since x and y are arbitrary integers, it follows that the sum of any two odd intgers is even. QED.

Now we'll write a more formal version of this proof, where odd(x) means x is odd and even(x) means x is even.

Proof:	1.	$\text{odd}(x)$	P
	2.	$\text{odd}(y)$	P
	3.	$\exists z\ (x = 2z + 1)$	1, definition of odd
	4.	$\exists z\ (y = 2z + 1)$	2, definition of odd
	5.	$x = 2m + 1$	3, EI
	6.	$y = 2n + 1$	4, EI
	7.	$x + y = 2(m + n + 1)$	5, 6, algebra
	8.	$\exists z\ (x + y = 2z)$	7, EG
	9.	$\text{even}(x + y)$	8, definition of even
	10.	$\text{odd}(x) \wedge \text{odd}(y) \rightarrow \text{even}(x + y)$	1, 2, 9, CP
	11.	$\forall y\ (\text{odd}(x) \wedge \text{odd}(y) \rightarrow \text{even}(x + y))$	10, UG
	12.	$\forall x\ \forall y\ (\text{odd}(x) \wedge \text{odd}(y) \rightarrow \text{even}(x + y))$	11, UG.
		QED	

Notice that lines 1 through 9 are indented to show that they form the subproof of the statement "If x and y are odd, then $x + y$ is even."

end example

7.3.6 Summary of Quantifier Proof Rules

Let's begin the summary by mentioning, as we did in Chapter 6, the following important usage note for inference rules.

Don't apply inference rules to subexpressions of wffs.

In other words, you can't apply an inference rule to just part of a wff. You have to apply it to the whole wff and nothing but the whole wff. So when applying the quantifier rules, remember to make sure that the wff in the numerator of the rule matches the wff on the line that you intend to use.

Before we summarize the four quantifier proof rules, let's recall that the phrase, "t is free to replace x in $W(x)$," means that either no variable of t

occurs bound to a quantifier in $W(x)$ or x does not occur free within the scope of a quantifier in $W(x)$. Equivalently, we can say that a term t is free to replace x in $W(x)$ if both $W(t)$ and $W(x)$ have the same bound occurrences of variables.

Universal Instantiation Rule (UI)	
$\dfrac{\forall x\ W(x)}{\therefore W(t)}$	*Restriction:* t is free to replace x in W(x).
Existential Generalization Rule (EG)	
$\dfrac{W(t)}{\therefore \exists x\ W(x)}$	*Restriction:* t is free to replace x in $W(x)$.
Existential Instantiation Rule (EI)	
$\dfrac{\exists x\ W(x)}{\therefore W(c)}$	*Restrictions:* c is a new constant in the proof and c does not occur in the conclusion of the proof.
Universal Generalization Rule (UG)	
$\dfrac{W(x)}{\therefore \forall x\ W(x)}$	*Restrictions:* Among the wffs used to infer $W(x), x$ is not free in any premise and x is not free in any wff constructed by EI.
Special Cases Where Restrictions are Always True	
$\dfrac{\forall x\ W(x)}{\therefore W(x)}$	$\dfrac{\forall x\ W(x)}{\therefore W(c)}$ (for any constant c)
$\dfrac{W(x)}{\therefore \exists x\ W(x)}$	$\dfrac{W(c)}{\therefore \exists x\ W(x)}$ (for any constant c)

Exercises

Restrictions using Quantifiers

1. Each of the following proof segments contains an invalid use of a quantifier proof rule. In each case, state why the proof rule cannot be used.

a. 1. $x < 4$ $\quad$ P
   2. $\forall x\ (x < 4)$ $\quad$ 1, UG.

b.	1.	$\exists x\ (y < x)$	P
	2.	$y < c$	1, EI
	3.	$\forall y\ (y < c)$	2, UG.
c.	1.	$\forall y\ (y < f(y))$	P
	2.	$\exists x\ \forall y\ (y < x)$	1, EG.
d.	1.	$q(x, c)$	P
	2.	$\exists x\ q(x, x)$	1, EG.
e.	1.	$\exists x\ p(x)$	P
	2.	$\exists x\ q(x)$	P
	3.	$p(c)$	1, EI
	4.	$q(c)$	2, EI.
f.	1.	$\forall x\ \exists y\ x < y$	P
	2.	$\exists y\ y < y$	1, UI.

2. Let W be the wff $\forall x\ (p(x) \vee q(x)) \rightarrow \forall x\ p(x) \vee \forall x\ q(x)$. It's easy to see that W is not valid. For example, let $p(x)$ mean "x is odd" and $q(x)$ mean "x is even" over the domain of integers. Then the antecedent is true, and the consequent is false. Suppose someone claims that the following sequence of statements is a "proof" of W:

1.	$\forall x\ (p(x) \vee q(x))$	P
2.	$p(x) \vee q(x)$	1, UI
3.	$\forall x\ p(x) \vee q(x)$	2, UG
4.	$\forall x\ p(x) \vee \forall x\ q(x)$	3, UG
	QED	1, 4, CP.

What is wrong with this "proof" of W?

3. a. Find a countermodel to show that the following wff is not valid:

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

b. The following argument attempts to prove that the wff in part (a) is valid. Find an error in the argument.

1.	$\exists x\ P(x)$	P
2.	$P(d)$	1, EI
3.	$\exists x\ (P(x) \rightarrow Q(x))$	P
4.	$P(d) \rightarrow Q(d)$	3, EI
5.	$Q(d)$	2, 4, MP
6.	$\exists x\ Q(x)$	5, EG.

4. Explain what is wrong with the following attempted proof.

1.	$p(x)$	P
2.	$\forall x\ q(x)$	P
3.	$q(x)$	2, UI
4.	$p(x) \wedge q(x)$	1, 3, Conj
5.	$\forall x\ (p(x) \wedge q(x))$	4, UG.
	QED?	

5. We'll give a formal proof of the following statement.

$$\forall x\ (p(x) \rightarrow q(x) \vee p(x)).$$

Proof:	1.	$p(x)$	P
	2.	$q(x) \vee p(x)$	1, Add
	3.	$p(x) \rightarrow q(x) \vee p(x)$	1, 2, CP
	4.	$\forall x\ (p(x) \rightarrow q(x) \vee p(x))$	3, UG
		QED.	

Suppose someone argues against this proof as follows: The variable x is free in the premise on line 1, which is used to infer line 3, so we can't use UG to generalize the wff on line 3. What is wrong with this argument?

Direct Proofs

6. Use the CP rule to prove that each of the following wffs is valid.
   a. $\forall x\ p(x) \rightarrow \exists x\ p(x)$.
   b. $\forall x\ (p(x) \rightarrow q(x)) \wedge \exists x\ p(x) \rightarrow \exists x\ q(x)$.
   c. $\exists x\ (p(x) \wedge q(x)) \rightarrow \exists x\ p(x) \wedge \exists x\ q(x)$.
   d. $\forall x\ (p(x) \rightarrow q(x)) \rightarrow (\exists x\ p(x) \rightarrow \exists x\ q(x))$.
   e. $\forall x\ (p(x) \rightarrow q(x)) \rightarrow (\forall x\ p(x) \rightarrow \exists x\ q(x))$.
   f. $\forall x\ (p(x) \rightarrow q(x)) \rightarrow (\forall x\ p(x) \rightarrow \forall x\ q(x))$.
   g. $\exists y\ \forall x\ p(x, y) \rightarrow \forall x\ \exists y\ p(x, y)$.
   h. $\exists x\ \forall y\ p(x, y) \wedge \forall x\ (p(x, x) \rightarrow \exists y\ q(y, x)) \rightarrow \exists y\ \exists x\ q(x, y)$.

Indirect Proofs

7. Use the IP rule to prove each that each of the following wffs is valid.
   a. $\forall x\ p(x) \rightarrow \exists x\ p(x)$.
   b. $\forall x\ (p(x) \rightarrow q(x)) \wedge \exists x\ p(x) \rightarrow \exists x\ q(x)$.
   c. $\exists y\ \forall x\ p(x, y) \rightarrow \forall x\ \exists y\ p(x, y)$.
   d. $\exists x\ \forall y\ p(x, y) \wedge \forall x\ (p(x, x) \rightarrow \exists y\ q(y, x)) \rightarrow \exists y\ \exists x\ q(x, y)$.
   e. $\forall x\ p(x) \vee \forall x\ q(x) \rightarrow \forall x\ (p(x) \vee q(x))$.

Transforming English Arguments

8. Transform each informal argument into a formalized wff. Then give a formal proof of the wff, using either CP or IP.
   a. Every dog either likes people or hates cats. Rover is a dog. Rover loves cats. Therefore, some dog likes people.
   b. Every committee member is rich and famous. Some committee members are old. Therefore, some committee members are old and famous.
   c. No human beings are quadrupeds. All men are human beings. Therefore, no man is a quadruped.
   d. Every rational number is a real number. There is a rational number. Therefore, there is a real number.
   e. Some freshmen like all sophomores. No freshman likes any junior. Therefore, no sophomore is a junior.

Equivalences

9. Give a formal proof for each of the following equivalences as follows: To prove $W \equiv V$, prove the two statements $W \rightarrow V$ and $V \rightarrow W$. Use either CP or IP.
   a. $\exists x\ \exists y\ W(x, y) \equiv \exists y\ \exists x\ W(x, y)$.
   b. $\forall x\ (A(x) \wedge B(x)) \equiv \forall x\ A(x) \wedge \forall x\ B(x)$.
   c. $\exists x\ (A(x) \vee B(x)) \equiv \exists x\ A(x) \vee \exists x\ B(x)$.
   d. $\exists x\ (A(x) \rightarrow B(x)) \equiv \forall x\ A(x) \rightarrow \exists x\ B(x)$.

Challenges

10. Give a formal proof of $A \rightarrow B$, where A and B are defined as follows:

$$A = \forall x\, (\exists y\, (q\,(x,y) \wedge s\,(y)) \rightarrow \exists y\, (p\,(y) \wedge r\,(x,y))),$$
$$B = \neg\, \exists x\ p\,(x) \rightarrow \forall x\ \forall y\, (q\,(x,y) \rightarrow \neg\, s\,(y)).$$

11. Give a formal proof of $A \rightarrow B$, where A and B are defined as follows:
    $A = \exists x\ (r(x) \wedge \forall y\ (p(y) \rightarrow q(x, y))) \wedge \forall x\ (r(x) \rightarrow \forall\ y\ (s(y) \rightarrow \neg\ q(x, y)))$,
    $B = \forall x\ (p(x) \rightarrow \neg\ s(x))$.

12. Each of the following wffs is *invalid*. Nevertheless, for each wff you are to construct a proof sequence that claims to be a proof of the wff but that fails because of the improper use of one or more inference rules. Also indicate which rules you use improperly and why the use is improper.
    a. $\exists x\ A(x) \rightarrow \forall x\ A(x)$.
    b. $\exists x\ A(x) \wedge \exists x\ B(x) \rightarrow \exists x\ (A(x) \wedge B(x))$.
    c. $\forall x\ (A(x) \vee B(x)) \rightarrow \forall x\ A(x) \vee \forall x\ B(x)$.
    d. $(\forall x\ A(x) \rightarrow \forall x\ B(x)) \rightarrow \forall x\ (A(x) \rightarrow B(x))$.
    e. $\forall x\ \exists y\ W(x, y) \rightarrow \exists y\ \forall x\ W(x, y)$.

13. Assume that x does not occur free in the wff C. Use either CP or IP to give a formal proof for each of the following equivalences.

   a. $\forall x\ (C \wedge A(x)) \equiv C \wedge \forall x\ A(x)$.
   b. $\exists x\ (C \wedge A(x)) \equiv C \wedge \exists x\ A(x)$.
   c. $\forall x\ (C \vee A(x)) \equiv C \vee \forall x\ A(x)$.
   d. $\exists x\ (C \vee A(x)) \equiv C \vee \exists x\ A(x)$.
   e. $\forall x\ (C \rightarrow A(x)) \equiv C \rightarrow \forall x\ A(x)$.
   f. $\exists x\ (C \rightarrow A(x)) \equiv C \rightarrow \exists x\ A(x)$.
   g. $\forall x\ (A(x) \rightarrow C) \equiv \exists x\ A(x) \rightarrow C$.
   h. $\exists x\ (A(x) \rightarrow C) \equiv \forall x\ A(x) \rightarrow C$.

14. Any inference rule for the propositional calculus can be converted to an inference rule for the predicate calculus. In other words, suppose R is an inference rule for the propositional calculus. If the hypotheses of R are valid wffs, then the conclusion of R is a valid wff. Prove this statement for each of the following inference rules.

   a. Modus tollens.
   b. Hypothetical syllogism.

15. Let $W(x)$ be a wff and t a term to be substituted for x in $W(x)$.

   a. Suppose t is free to replace x in $W(x)$ and there is a variable in t that is bound in $W(x)$. What can you say?
   b. Suppose no variable of t is bound in $W(x)$. What can you say?

16. If the term t is free to replace x in $W(x)$ and I is an interpretation, then $W(t)I = W(tI)I$. Prove the statement by induction on the number of connectives and quantifiers.

17. Any binary relation that is irreflexive and transitive is also antisymmetric. Here is an informal proof. Let p be a binary relation on a set A such that p is irreflexive and transitive. Suppose, by way of contradiction, that p is not antisymmetric. Then there are elements $a, b \in A$ such that $p(a, b)$ and $p(b, a)$. Since p is transitive, it follows that $p(a, a)$. But this contradicts the fact that p is irreflexive. Therefore, p is antisymmetric. Give a formal proof of the statement, where the following wffs represent the three properties:

   Irreflexive: $\forall x \neg p(x, x)$.

   Transitive: $\forall x\ \forall y\ \forall z\ (p(x, y) \wedge p(y, z) \rightarrow p(x, z))$.

   Antisymmetric: $\forall x\ \forall\ y\ (p(x, y) \rightarrow \neg p(y, x))$.

7.4 Chapter Summary

The first-order predicate calculus extends propositional calculus by allowing wffs to contain predicates and quantifiers of variables. Meanings for these wffs are defined in terms of interpretations over nonempty sets called domains. A wff is valid if it's true for all possible interpretations. A wff is unsatisfiable if it's false for all possible interpretations.

There are basic equivalences that allow us to simplify and transform wffs into other wffs. We can use equivalences to transform any wff into a prenex DNF or prenex CNF. Equivalences can also be used to compare different formalizations of the same English sentence.

To decide whether a wff is valid, we can try to transform it into an equivalent wff that we know to be valid. But, in general, we must rely on some type of informal or formal reasoning. A formal reasoning system for the first-order predicate calculus can use all the rules and proof techniques of the propositional calculus. But we need four additional inference rules for the quantifiers: universal instantiation, existential instantiation, universal generalization, and existential generalization.

Notes

Now we have the basics of logic—the propositional calculus and the first-order predicate calculus. In Section 6.4 we introduced a formal axiom system for the propositional calculus and we observed that the system is complete, which means that every tautology can be proven as a theorem within the system.

It's nice to know that there is a similar statement for the predicate calculus, which is due to the logician and mathematician Kurt Gödel (1906–1978). Gödel showed that the first-order predicate calculus is complete. In other words, there are formal systems for the first-order predicate calculus such that every valid wff can be proven as a theorem. The formal system presented by Gödel [1930] used fewer axioms and fewer inference rules than the system that we've been using in this chapter.

chapter 8

Applied Logic

Once the people begin to reason,
all is lost.
—Voltaire (1694–1778)

When we reason, we usually do it in a particular domain of discourse. For example, we might reason about computer science, politics, mathematics, physics, automobiles, or cooking. But these domains are usually too large to do much reasoning. So we normally narrow our scope of thought and reason in domains such as imperative programming languages, international trade, plane geometry, optics, suspension systems, or pasta recipes.

No matter what the domain of discussion, we usually try to correctly apply inferences while we are reasoning. Since each of us has our own personal reasoning system, we sometimes find it difficult to understand one another. In an attempt to find common ground among the various ways that people reason, we introduced the propositional calculus and first-order predicate calculus. So we've looked at some formalizations of logic.

Can we go a step further and formalize the things that we talk about? Many subjects can be formalized by giving some axioms that define the properties of the objects being discussed. For example, when we reason about geometry, we make assumptions about points and lines. When we reason about automobile engines, we make certain assumptions about how they work. When we combine first-order predicate calculus with the formalization of some subject, we obtain a reasoning system called a *first-order theory.*

chapter guide

Section 8.1 shows how the fundamental notion of equality can become part of a first-order theory.

Section 8.2 introduces a first-order theory for proving the correctness of imperative programs.

Section 8.3 introduces logics that are beyond the first order. We'll give some examples to show how higher-order logics can be used to formalize much of our natural discourse.

8.1 Equality

Equality is a familiar notion to most of us. For example, we might compare two things to see whether they are equal, or we might replace a thing by an equal thing during some calculation. In fact, equality is so familiar that we might think that it does not need to be discussed further. But we are going to discuss it further because different domains of discourse often use equality in different ways. If we want to formalize some subject that uses the notion of equality, then it should be helpful to know basic properties that are common to all equalities.

A first-order theory is called a *first-order theory with equality* if it contains a two-argument predicate, say e, that captures the properties of equality required by the theory. We usually denote $e(x, y)$ by the familiar

$$x = y.$$

Similarly, we let $x \neq y$ denote $\neg\, e(x, y)$.

Let's examine how we use equality in our daily discourse. We always assume that any term is equal to itself. For example, $x = x$ and $f(c) = f(c)$. We might call this "syntactic equality."

Another familiar use of equality might be called "semantic equality." For example, although the expressions $2 + 3$ and $1 + 4$ are not syntactically equal, we still write $2 + 3 = 1 + 4$ because they both represent the same number.

Another important use of equality is to replace equals for equals in an expression. The following examples should get the point across.

$$\begin{aligned}
&\text{If } x + y = 2z, \text{ then } (x + y) + w = 2z + w.\\
&\text{If } x = y, \text{ then } f(x) = f(y).\\
&\text{If } f(x) = f(y), \text{ then } g(f(x)) = g(f(y)).\\
&\text{If } x = y + z, \text{ then } 8 < x \equiv 8 < y + z.\\
&\text{If } x = y, \text{ then } p(x) \vee q(w) \equiv p(y) \vee q(w).
\end{aligned}$$

8.1.1 Describing Equality

Let's try to describe some fundamental properties that all first-order theories with equality should satisfy. Of course, we want equality to satisfy the basic property that each term is equal to itself. The following axiom will suffice for this purpose.

Equality Axiom (EA) **(8.1)**

$$\forall x\ (x = x).$$

This axiom tells us that $x = x$ for all variables x. The axiom is sometimes called the *law of identity*. But we also want to say that $t = t$ for any term t. For example, if a theory contains a term such as $f(x)$, we certainly want to say that $f(x) = f(x)$. Do we need another axiom to tell us that each term is equal to itself? No. All we need is a little proof sequence as follows:

1.	$\forall x\ (x = x)$	EA
2.	$t = t$	1, UI.

So for any term t we have $t = t$. Because this is such a useful result, we'll also refer to it as EA. In other words, we have

Equality Axiom (EA) **(8.2)**

$t = t$ for all terms t.

Now let's try to describe that well-known piece of folklore, *equals can replace equals*. Since this idea has such a wide variety of uses, it's hard to tell where to begin. So we'll start with a rule that describes the process of replacing some occurrence of a term in a predicate by an equal term. In this rule, p denotes an arbitrary predicate with one or more arguments. The letters t and u represent arbitrary terms.

Equals-for-Equals Rule (EE) **(8.3)**

$$(t = u) \wedge p(\ldots\ t\ \ldots) \rightarrow p(\ldots\ u\ \ldots).$$

The notations $\ldots\ t\ \ldots$ and $\ldots\ u\ \ldots$ indicate that t and u occur in the same argument place of p. In other words, u replaces the indicated occurrence of t. Since (8.3) is an implication, we can use it as an inference rule in the following equivalent form.

Equals-for-Equals Rule (EE) **(8.4)**

$$\frac{t = u,\ p(\ldots t \ldots)}{\therefore p(\ldots u \ldots)}.$$

The EE rule is sometimes called the *principle of extensionality.* Let's see what we can conclude from EE. Whenever we discuss equality of terms, we usually want the following two properties to hold for all terms:

Symmetric: $(t = u) \to (u = t)$.

Transitive: $(t = u) \wedge (u = v) \to (t = v)$.

We'll use the EE rule to prove the symmetric property in the next example and leave the transitive property as an exercise.

example 8.1 A Proof of Symmetry

We'll prove the symmetric property $(t = u) \to (u = t)$.

Proof:			
	1.	$t = u$	P
	2.	$t = t$	EA
	3.	$u = t$	1, 2, EE
		QED	1, 3, CP.

To see why the statement on line 3 follows from the EE rule, we'll let $p(x, y)$ mean "$x = y$." Then the proof can be rewritten in terms of p as follows:

Proof:			
	1.	$t = u$	P
	2.	$p(t, t)$	EA
	3.	$p(u, t)$	1, 2, EE
		QED	1, 3, CP.

end example

Another thing we would like to conclude from EE is that equals can replace equals in a term like $f(\dots\ t \dots)$. In other words, we would like the following wff to be valid:

$$(t = u) \to f(\dots\ t \dots) = f(\dots\ u \dots).$$

To prove that this wff is valid, we'll let $p(t, u)$ mean "$f(\dots\ t \dots) = f(\dots\ u \dots)$." Then the proof goes as follows:

Proof:			
	1.	$t = u$	P
	2.	$p(t, t)$	EA
	3.	$p(t, u)$	1, 2, EE
		QED	1, 3, CP.

When we're dealing with axioms for a theory, we sometimes write down more axioms than we really need. For example, some axiom might be deducible as a theorem from the other axioms. The practical purpose for this is to have a listing of the useful properties all in one place. For example, to describe equality for terms, we might write down the following five statements as axioms.

Equality Axioms for Terms **(8.5)**

In these axioms the letters t, u, and v denote arbitrary terms, f is an arbitrary function, and p is an arbitrary predicate.

EA:	$t = t.$
Symmetric:	$(t = u) \to (u = t)\,.$
Transitive:	$(t = u) \wedge (u = v) \to (t = v)\,.$
EE (functional form):	$(t = u) \to f(\ldots t \ldots) = f(\ldots u \ldots)\,.$
EE (predicate form):	$(t = u) \wedge p(\ldots t \ldots) \to p(\ldots u \ldots)\,.$

The EE axioms in (8.5) allow only a single occurrence of t to be replaced by u. We may want to substitute more than one "equals for equals" at the same time. For example, if $x = a$ and $y = b$, we would like to say that $f(x, y) = f(a, b)$. It's nice to know that simultaneous use of equals for equals can be deduced from the axioms. For example, we'll prove the following statement:

$$(x = a) \wedge (y = b) \to f(x, y) = f(a, b)\,.$$

Proof:	1.	$x = a$	P
	2.	$y = b$	P
	3.	$f(x, y) = f(a, y)$	1, EE
	4.	$f(a, y) = f(a, b)$	2, EE
	5.	$f(x, y) = f(a, b)$	3, 4, Transitive
		QED	1, 2, 5, CP.

This proof can be extended to substitute any number of equals for equals simultaneously in a function or in a predicate. In other words, we could have written the two EE axioms of (8.5) in the following form.

Multiple Replacement EE **(8.6)**

In these axioms the letters t, u, and v denote arbitrary terms, f is an arbitrary function, and p is an arbitrary predicate.

EE (function): $(t_1 = u_1) \wedge \cdots \wedge (t_k = u_k) \to f(t_1, \ldots, t_k) = f(u_1, \ldots, u_k)$.

EE (predicate): $(t_1 = u_1) \wedge \cdots \wedge (t_k = u_k) \wedge p(t_1, \ldots, t_k) = p(u_1, \ldots, u_k)$.

So the two axioms (8.1) and (8.3) are sufficient for us to deduce all the axioms in (8.5) together with those of (8.6).

Working with Numeric Expressions

Let's consider the set of arithmetic expressions over the domain of integers, together with the usual arithmetic operations. The terms in this theory are arithmetic expressions, such as

$$35, \quad x, \quad 2 + 8, \quad x + y, \quad 6x - 5 + y.$$

Equality of terms comes into play when we write statements such as

$$3 + 6 = 2 + 7, \quad 4 \neq 2 + 3.$$

We have axioms to tell how the operations work. For example, we know that the + operation is associative, and we know that $x + 0 = x$ and $x - x = 0$. We can reason in such a theory by using the predicate calculus with equality. In the next two examples we'll give an informal proof and a formal proof of the following well-known statement:

$$\forall x\ ((x + x = x) \rightarrow (x = 0)).$$

example

8.2 An Informal Proof

First we'll do an informal equational type proof. Let x be any number such that $x + x = x$. Then we have the following equations:

$x = x + 0$	(property of 0)
$= x + (x + -x)$	(property of $-$)
$= (x + x) + -x$	(associativity of $+$)
$= x + -x$	(hypothesis $x + x = x$)
$= 0$	(property of $-$)

Since x was arbitary, the statement is true for all x. QED.

end example

example

8.3 A Formal Proof

In the informal proof we used several instances of equals for equals. Now let's look at a formal proof in all its glory.

Proof:	1.	$x + x = x$	P
	2.	$-x = -x$	EA
	3.	$(x + x) + -x = x + -x$	1, 2, EE
	4.	$x + (x + -x) = (x + x) + -x$	Associativity
	5.	$x + (x + -x) = x + -x$	3, 4, Transitivity

6.	$x + -x = 0$	Property of –
7.	$x + 0 = 0$	5, 6, EE
8.	$x = x + 0$	Property of 0
9.	$x = 0$	7, 8, Transitivity
10.	$(x + x = x) \rightarrow (x = 0)$	1, 9, CP
11.	$\forall x\ ((x + x = x) \rightarrow (x = 0))$	10, UG.
	QED	

Let's explain the two uses of EE. For line 3, let $f(u, v) = u + v$. Then the wff on line 3 results from lines 1 and 2 together with the following instance of EE in functional form:

$$(x + x = x) \rightarrow f(x + x, -x) = f(x, -x).$$

For line 7, let $p(u, v)$ denote the statement "$u + v = v$." Then the wff on line 7 results from lines 5 and 6 together with the following instance of EE in predicate form:

$$(x + -x = 0) \wedge p(x, x + -x) \rightarrow p(x, 0).$$

end example

Partial Order Theories

A *partial order theory* is a first-order theory with equality that also contains an ordering predicate that is antisymmetric and transitive. If the ordering predicate is reflexive, we denote it by $\leq$. If it is irreflexive, we denote it by $<$.

For example, the antisymmetric and transitive properties for $\leq$ can be written as follows, where x, y, and z are arbitrary elements.

Antisymmetric: $(x \leq y) \wedge (y \leq x) \rightarrow (x = y)$.
Transitive: $(x \leq y) \wedge (y \leq z) \rightarrow (x \leq z)$.

We can use equality to define either one of the relations $<$ and $\leq$ in terms of the other in the following way.

$$x < y \quad \text{means} \quad (x \leq y) \wedge (x \neq y)\,,$$
$$x \leq y \quad \text{means} \quad (x < y) \vee (x = y)\,.$$

We can do formal reasoning in such a first-order theory in much the same way that we reason informally.

example

8.4 An Obvious Statement

Most of us use the following statement without even thinking:

$$(x < y) \rightarrow (x \leq y).$$

Here are two different proofs of the statement.

Proof:				
	1.	$x < y$	P	
	2.	$(x \leq y) \wedge (x \neq y)$	1, T	(definition)
	3.	$x \leq y$	2, Simp	
	4.	$(x < y) \to (x \leq y)$	1, 3, CP.	
		QED		

Proof:				
	1.	$x < y$	P	
	2.	$(x < y) \vee (x = y)$	1, Addition	
	3.	$x \leq y$	2, T	(definition)
	4.	$(x < y) \to (x \leq y)$	1, 3, CP.	
		QED		

end example

8.1.2 Extending Equals for Equals

The EE rule for replacing equals for equals in a predicate can be extended to other wffs. For example, we can use the EE rule to prove the following more general statement about wffs without quantifiers.

EE for Wffs with no Quantifiers **(8.7)**

If $W(x)$ has no quantifiers, then the following wff is valid:

$$(t = u) \wedge W(t) \to W(u).$$

We assume that $W(t)$ is obtained from $W(x)$ by replacing one or more occurrences of x by t and that $W(u)$ is obtained from $W(t)$ by replacing one or more occurrences of t by u.

For example, if $W(x) = p(x, y) \wedge q(x, x)$, then we might have $W(t) = p(t, y) \wedge q(x, t)$, where only two of the three occurrences of x are replaced by t. In this case we might have $W(u) = p(u, y) \wedge q(x, t)$, where only one occurrence of t is replaced by u. In other words, the following wff is valid:

$$(t = u) \wedge p(t, y) \wedge q(x, t) \to p(u, y) \wedge q(x, t).$$

What about wffs that contain quantifiers? Even when a wff has quantifiers, we can use the EE rule if we are careful not to introduce new bound occurrences of variables. Here is the full-blown version of EE.

EE for Wffs with Quantifiers (8.8)

If $W(x)$ is a wff and t and u are terms that are free to replace x in $W(x)$, then the following wff is valid:

$$(t = u) \wedge W(t) \rightarrow W(u).$$

We assume that $W(t)$ is obtained from $W(x)$ by replacing one or more occurrences of x by t and that $W(u)$ is obtained from $W(t)$ by replacing one or more occurrences of t by u.

For example, suppose $W(x) = \exists y\ p(x, y)$. Then for any terms t and u that do not contain occurrences of y, the following wff is valid:

$$(t = u) \wedge \exists y\ p(t, y) \rightarrow \exists y\ p(u, y).$$

The exercises contain some samples to show how EE for predicates (8.3) can be used to prove some simple extensions of EE to more general wffs.

Exercises

Equals for Equals

1. Use the EE rule to prove the double replacement rule:

$$(s = v) \wedge (t = w) \wedge p(s, t) \rightarrow p(v, w).$$

2. Show that the transitive property $(t = u) \wedge (u = v) \rightarrow (t = v)$ can be deduced from the other axioms for equality (8.5).

3. Give a formal proof of the following statement about the integers:

$$(c = a^i) \wedge (i \leq b) \wedge \neg (i < b) \rightarrow (c = a^b).$$

4. Use the equality axioms (8.5) to prove each of the following versions of EE, where p and q are predicates, t and u are terms, and x, y, and z are variables.
   a. $(t = u) \wedge \neg p(\ldots\ t \ldots) \rightarrow \neg p(\ldots\ u \ldots)$.
   b. $(t = u) \wedge p(\ldots\ t \ldots) \wedge q(\ldots\ t \ldots) \rightarrow p(\ldots\ u \ldots) \wedge q(\ldots\ u \ldots)$.
   c. $(t = u) \wedge (p(\ldots\ t \ldots) \vee q(\ldots\ t \ldots)) \rightarrow p(\ldots\ u \ldots) \vee q(\ldots\ u \ldots)$.
   d. $(x = y) \wedge \exists z\ p(\ldots\ x \ldots) \rightarrow \exists z\ p(\ldots\ y \ldots)$.
   e. $(x = y) \wedge \forall z\ p(\ldots\ x \ldots) \rightarrow \forall z\ p(\ldots\ y \ldots)$.

5. Prove the validity of the wff $\forall x\ \exists y\ (x = y)$.

6. Prove each of the following equivalences.

   a. $p(x) \equiv \exists y\ ((x = y) \wedge p(y))$.
   b. $p(x) \equiv \forall y\ ((x = y) \rightarrow p(y))$.

Formalizing English Sentences

7. Formalize the definition for each statement about the integers.

   a. odd(x) means x is odd.
   b. even(x) means x is even.
   c. div(a, b) means a divides b.
   d. $r = a \bmod b$.
   e. $d = \gcd(a, b)$.

8. Formalize each of the following statements.

   a. There is at most one x such that $A(x)$ is true.
   b. There are exactly two x and y such that that $A(x)$ and $A(y)$ are true.
   c. There are at most two x and y such that $A(x)$ and $A(y)$ are true.

9. Students were asked to formalize the statement "There is a unique x such that $A(x)$ is true." The following wffs were given as answers.

   a. $\exists x\ (A(x) \wedge \forall y\ (A(y) \rightarrow (x = y)))$.
   b. $\exists x\ A(x) \wedge \forall x\ \forall y\ (A(x) \wedge A(y) \rightarrow (x = y))$.

   Prove that wffs (a) and (b) are equivalent. *Hint:* Do two indirect proofs showing that each statement implies the other.

8.2 Program Correctness

An important and difficult problem of computer science can be stated as

$$\text{"Prove that a program is correct."} \qquad (8.9)$$

This takes some discussion. One major question to ask before we can prove that a program is correct is "What is the program supposed to do?" If we can state in English what a program is supposed to do, and English is the programming language, then the statement of the problem may itself be a proof of its correctness.

Normally, a problem is stated in some language X, and its solution is given in some language Y. For example, the statement of the problem might use English mixed with some symbolic notation, while the solution might be in a programming language. How do we prove correctness in cases like this? Often the answer depends on the programming language. As an example, we'll look at a formal theory for proving the correctness of imperative programs.

8.2.1 Imperative Program Correctness

An *imperative program* consists of a sequence of statements that represent commands. The most important statement is the assignment statement. Other statements are used for control, such as looping and taking alternate paths. To prove things about such programs, we need a formal theory consisting of wffs, axioms, and inference rules.

Suppose we want to prove that a program does some particular thing. We must represent the thing that we want to prove in terms of a precondition P, which states what is supposed to be true before the program starts, and a postcondition Q, which states what is supposed to be true after the program halts. If S denotes the program, then we will describe this informal situation with the following wff, which is called a *Hoare triple:*

$$\{P\}\ S\ \{Q\}.$$

The letters P and Q denote logical statements that describe properties of the variables that occur in S. P is called a *precondition* for S, and Q is called a *postcondition* for S. We assume that P and Q are wffs from a first-order theory with equality that depends on the program S. For example, if the program manipulates numbers, then the first-order theory must include the numerical operations and properties that are required to describe the problem at hand. If the program processes strings, then the first-order theory must include the string operations.

For example, suppose S is the single assignment statement $x := x + 1$. Then the following expression is a wff in our logic:

$$\{x > 4\}\ x := x + 1\ \{x > 5\}.$$

To have a logic for program correctness, we must have a meaning assigned to each wff of the form $\{P\}\ S\ \{Q\}$. In other words, we need to assign a truth value to each wff of the form $\{P\}\ S\ \{Q\}$.

The Meaning of $\{P\}\ S\ \{Q\}$

The meaning of $\{P\}\ S\ \{Q\}$ is the truth value of the following statement:

> If P is true before S is executed and the execution of S terminates, then Q is true after the execution of S.

If $\{P\}\ S\ \{Q\}$ is true, we say S is *correct* with respect to precondition P and postcondition Q. Strictly speaking, we should say that S is *partially correct* because the truth of $\{P\}\ S\ \{Q\}$ is based on the assumption that S terminates. If we also know that S terminates, then we say S is *totally correct.* We'll discuss termination at the end of the section.

Sometimes it's easy to observe whether $\{P\}\ S\ \{Q\}$ is true. For example, from our knowledge of the assignment statement, most of us will agree that the following wff is true:

$$\{x > 4\}\ x := x + 1\ \{x > 5\}.$$

On the other hand, most of us will also agree that the following wff is false:

$$\{x > 4\}\ x := x + 1\ \{x > 6\}.$$

But we need some proof methods to verify our intuition. A formal theory for proving the correctness of programs needs some axioms and some inference rules. So let's start.

The Assignment Axiom

The axioms depend on the types of assignments allowed by the assignment statement. The inference rules depend on the control structures of the language. So we had better agree on a language before we go any further in our discussion. To keep things simple, we'll assume that the assignment statement has the following form, where x is a variable and t is a term:

$$x := t.$$

So the only thing we can do is assign a value to a variable. This effectively restricts the language so that it cannot use other structures, such as arrays and records. In other words, we can't make assignments like $a[i] := t$ or $a.b := t$.

Since our assignment statement is restricted to the form $x := t$, we need only one axiom. It's called the *assignment axiom*, and we'll motivate the discovery of the axiom by an example. Suppose we're told that the following wff is correct:

$$\{P\}\ x := 4\ \{y > x\}.$$

In other words, if P is true before the execution of the assignment statement, then after its execution the statement $y > x$ is true. What should P be? From our knowledge of the assignment statement we might guess that P has the following definition:

$$P = (y > 4).$$

This is about the most general statement we can make. Notice that P can be obtained from the postcondition $y > x$ by replacing x by 4. The assignment axiom generalizes this idea in the following way.

Assignment Axiom (AA) **(8.10)**

$$\{Q(x/t)\}\ x := t\ \{Q\}.$$

The notation $Q(x/t)$ denotes the wff obtained from Q by replacing all free occurrences of x by t. The axiom is often called the "backwards" assignment axiom because the precondition is constructed from the postcondition.

Let's see how the assignment axiom works in a backwards manner. When using AA, always start by writing down the form of (8.10) with an empty precondition as follows:

$$\{\quad\}\ x := t\ \{Q\}.$$

Now the task is to construct the precondition by replacing all free occurrences of x in Q by t.

For example, suppose we know that $x < 5$ is the postcondition for the assignment statement $x := x + 1$. We start by writing down the following partially completed version of AA:

$$\{\quad\}\ x := x + 1\ \{x < 5\}.$$

Then we use AA to construct the precondition. In this case we replace the x by $x + 1$ in the postcondition $x < 5$. This gives us the precondition $x + 1 < 5$, and we can write down the completed instance of the assignment axiom:

$$\{x + 1 < 5\}\ x := x + 1\ \{x < 5\}.$$

The Consequence Rule

It happens quite often that the precondition constructed by AA doesn't quite match what we're looking for. For example, most of us will agree that the following wff is correct.

$$\{x < 3\}\ x := x + 1\ \{x < 5\}.$$

But we've already seen that AA applied to this assignment statement gives

$$\{x + 1 < 5\}\ x := x + 1\ \{x < 5\}.$$

Since the two preconditions don't match, we have some more work to do. In this case we know that for any number x we have $(x < 3) \rightarrow (x + 1 < 5)$.

Let's see why this is enough to prove that $\{x < 3\}\ x := x + 1\ \{x < 5\}$ is correct. If $x < 3$ is true before the execution of $x := x + 1$, then we also know that $x + 1 < 5$ is true before execution of $x := x + 1$. Now AA tells us that $x < 5$ is true after execution of $x := x + 1$. So $\{x < 3\}\ x := x + 1\ \{x < 5\}$ is correct.

This kind of argument happens so often that we have an inference rule to describe the situation for any program S. It's called the *consequence rule:*

Consequence Rules **(8.11)**

$$\frac{P \rightarrow R \text{ and } \{R\}\,S\,\{Q\}}{\therefore \{P\}\,S\,\{Q\}} \quad \text{and} \quad \frac{\{P\}\,S\,\{T\} \text{ and } T \rightarrow Q,}{\therefore \{P\}\,S\,\{Q\}}.$$

Notice that each consequence rule requires two proofs: a proof of correctness and a proof of an implication. Let's do an example.

example

8.5 Using the Assignment Axiom and the Consequence Rule

We'll prove the correctness of the following wff:

$$\{x < 5\}\ x := x + 1\ \{x < 7\}.$$

To start things off, we'll apply (8.10) to the assignment statement and the postcondition to obtain the following wff:

$$\{x + 1 < 7\}\ x := x + 1\ \{x < 7\}.$$

This isn't what we want. We got the precondition $x + 1 < 7$, but we need the precondition $x < 5$. Let's see whether we can apply (8.11) to the problem. In other words, let's see whether we can prove the following statement:

$$(x < 5) \rightarrow (x + 1 < 7).$$

This statement is certainly true, and we'll include its proof in the following formal proof of correctness of the original wff.

Proof:	1.	$\{x + 1 < 7\}\ x := x + 1\ \{x < 7\}$	AA
	2.	$x < 5$	P
	3.	$x + 1 < 6$	2, T
	4.	$6 < 7$	T
	5.	$x + 1 < 7$	3, 4, Transitive
	6.	$(x < 5) \rightarrow (x + 1 < 7)$	2, 5, CP
		QED	1, 6, Consequence.

end example

Although assignment statements are the core of imperative programming, we can't do much programming without control structures. So let's look at a few fundamental control structures together with their corresponding inference rules.

The Composition Rule

The most basic control structure is the composition of two statements S_1 and S_2, which we denote by S_1; S_2. This means execute S_1 and then execute S_2. The *composition rule* can be used to prove the correctness of the composition of two statements.

Composition Rule **(8.12)**

$$\frac{\{P\}\,S_1\,\{R\} \text{ and } \{R\}\,S_2\,\{Q\}}{\therefore \{P\}\,S_1; S_2\,\{Q\}}.$$

The composition rule extends naturally to any number of program statements in a sequence. For example, suppose we prove that the following three wffs are correct.

$$\{P\}\ S_1\ \{R\},\quad \{R\}\ S_2\ \{T\},\quad \{T\}\ S_3\ \{Q\}.$$

Then we can infer that $\{P\}\ S_1;\ S_2;\ S_3\ \{Q\}$ is correct.

For (8.12) to work, we need an intermediate condition R to place between the two statements. Intermediate conditions often appear naturally during a proof, as the next example shows.

example 8.6 Using the Composition Rule

We'll show the correctness of the following wff:

$$\{(x > 2) \wedge (y > 3)\}\ x := x + 1;\ y := y + x\ \{y > 6\}.$$

This wff matches the bottom of the composition inference rule (8.12). Since the program statements are assignments, we can use the AA rule to move backward from the postcondition to find an intermediate condition to place between the two assignments. Then we can use AA again to move backward from the intermediate condition. Here's the proof.

Proof: First we'll use AA to work backward from the postcondition through the second assignment statement:

1. $\{y + x > 6\}\ y := y + x\ \{y > 6\}$ AA

Now we can take the new precondition and use AA to work backward from it through the first assignment statement:

2. $\{y + x + 1 > 6\}\ x := x + 1\ \{y + x > 6\}$ AA

Now we can use the composition rule (8.12) together with lines 1 and 2 to obtain line 3 as follows:

3. $\{y + x + 1 > 6\}\ x := x + 1;\ y := y + x\ \{y > 6\}$ 1, 2, Comp

At this point the precondition on line 3 does not match the precondition for the wff that we are trying to prove correct. Let's try to apply the consequence rule (8.11) to the situation.

4.	$(x > 2) \wedge (y > 3)$	P
5.	$x > 2$	4, Simp
6.	$y > 3$	4, Simp
7.	$x + y > 2 + y$	5, T
8.	$2 + y > 2 + 3$	6, T
9.	$x + y > 2 + 3$	7, 8, Transitive
10.	$x + y + 1 > 6$	9, T
11.	$(x > 2) \wedge (y > 3) \rightarrow (x + y + 1 > 6)$	4, 10, CP

Now we're in position to apply the consequence rule to lines 3 and 11:

12. $\{(x > 2) \wedge (y > 3)\}\ x := x + 1;\ y := y + x\ \{y > 6\}$
3, 11, Consequence.

QED

end example

The If-Then Rule

The statement **if** C **then** S means that S is executed if C is true and S is bypassed if C is false. For statements of this form we have the following *if-then rule* of inference.

If-Then Rule **(8.13)**

$$\frac{\{P \wedge C\}\, S\, \{Q\} \text{ and } P \wedge \neg\, C \rightarrow Q}{\therefore \{P\}\, \textbf{if } C \textbf{ then } S\, \{Q\}}.$$

The two wffs in the hypothesis of (8.13) are of different type. The logical wff $P \wedge \neg\, C \rightarrow Q$ needs a proof from the predicate calculus. This wff is necessary in the hypothesis of (8.13) because if C is false, then S does not execute. But we still need Q to be true after C has been determined to be false during the execution of the if-then statement. Let's do an example.

example 8.7 Using the If-Then Rule

We'll show that the following wff is correct:

$$\{\text{true}\}\ \textbf{if } x < 0 \textbf{ then } x := -x\ \{x \geq 0\}.$$

Proof: Since the wff fits the pattern of (8.13), all we need to do is prove the following two statements:

1. $\{\text{true} \wedge (x < 0)\}\ x := -x\ \{x \geq 0\}$.
2. $\text{true} \wedge \neg\, (x < 0) \rightarrow (x \geq 0)$.

The proofs are easy. We'll combine them into one formal proof:

Proof:	1.	$\{-x \geq 0\}\ x := -x\ \{x \geq 0\}$	AA
	2.	$\quad\quad \text{true} \wedge (x < 0)$	P
	3.	$\quad\quad x < 0$	2, Simp
	4.	$\quad\quad -x > 0$	3, T
	5.	$\quad\quad -x \geq 0$	4, Add
	6.	$\text{true} \wedge (x < 0) \rightarrow (-x \geq 0)$	2, 5, CP
	7.	$\{\text{true} \wedge (x < 0)\}\ x := -x\ \{x \geq 0\}$	1, 6, Consequence
	8.	$\quad\quad \text{true} \wedge \neg\,(x < 0)$	P
	9.	$\quad\quad \neg\,(x < 0)$	8, Simp
	10.	$\quad\quad x \geq 0$	9, T
	11.	$\text{true} \wedge \neg\,(x < 0) \rightarrow (x \geq 0)$	8, 10, CP
		QED	7, 11, If-then.

end example

The If-Then-Else Rule

The statement **if** C **then** S_1 **else** S_2 means that S_1 is executed if C is true and S_2 is executed if C is false. For statements of this form we have the following *if-then-else rule* of inference.

If-Then-Else Rule **(8.14)**

$$\frac{\{P \wedge C\}\, S_1\, \{Q\} \text{ and } \{P \wedge \neg\, C\}\, S_2\, \{Q\}}{\therefore \{P\}\ \textbf{if}\ C\ \textbf{then}\ S_1\ \textbf{else}\ S_2\, \{Q\}}.$$

example

8.8 Using the If-Then-Else Rule

Suppose we're given the following wff, where even(x) means that x is an even integer:

$$\{\text{true}\}\ \textbf{if}\ \text{even}(x)\ \textbf{then}\ y := x\ \textbf{else}\ y := x + 1\ \{\text{even}(y)\}.$$

We'll give a formal proof that this wff is correct. The wff matches the bottom of rule (8.14). Therefore, the wff will be correct by (8.14) if we can show that the following two wffs are correct:

1. $\{\text{true} \wedge \text{even}(x)\}\ y := x\ \{\text{even}(y)\}$.
2. $\{\text{true} \wedge \text{odd}(x)\}\ y := x + 1\ \{\text{even}(y)\}$.

To make the proof formal, we need to give formal descriptions of even(x) and odd(x). This is easy to do over the domain of integers.

$$\text{even}(x) = \exists k\,(x = 2k),$$
$$\text{odd}(x) = \exists k\,(x = 2k + 1).$$

To avoid clutter, we'll use even(x) and odd(x) in place of the formal expressions. If you want to see why a particular line holds, you might make the substitution for even or odd and then see whether the statement makes sense. We'll combine the two proofs into the following formal proof:

Proof:			
	1.	$\{\text{even}(x)\}\ y := x\ \{\text{even}(y)\}$	AA
	2.	$\quad\quad \text{true} \wedge \text{even}(x)$	P
	3.	$\quad\quad \text{even}(x)$	2, Simp
	4.	$\text{true} \wedge \text{even}(x) \to \text{even}(x)$	2, 3, CP
	5.	$\{\text{true} \wedge \text{even}(x)\}\ y := x\ \{\text{even}(y)\}$	1, 4, Consequence
	6.	$\{\text{even}(x + 1)\}\ y := x + 1\ \{\text{even}(y)\}$	AA
	7.	$\quad\quad \text{true} \wedge \text{odd}(x)$	P
	8.	$\quad\quad \text{odd}(x)$	7, Simp
	9.	$\quad\quad \text{even}(x + 1)$	8, T
	10.	$\text{true} \wedge \text{odd}(x) \to \text{even}(x + 1)$	7, 9, CP
	11.	$\{\text{true} \wedge \text{odd}(x)\}\ y := x + 1\ \{\text{even}(y)\}$	6, 10, Consequence
		QED	5, 11, If-then-else.

end example

The While Rule

The last inference rule that we will consider is the *while rule.* The statement **while** C **do** S means that S is executed if C is true, and if C is still true after S has executed, then the process is started over again. Since the body S may execute more than once, there must be a close connection between the precondition and postcondition for S. This can be seen by the appearance of P in all preconditions and postconditions of the rule.

While Rule **(8.15)**

$$\frac{\{P \wedge C\}\, S\, \{P\}}{\therefore\ \{P\}\,\textbf{while}\ C\ \textbf{do}\ \ S\,\{P \wedge \neg\, C\}}.$$

The wff P is called a *loop invariant* because it must be true before and after each execution of the body S. Loop invariants can be tough to find in programs with no documentation. On the other hand, in writing a program, a loop invariant can be a helpful tool for specifying the actions of while loops.

To illustrate the idea of working with while loops, we'll work our way through an example that will force us to discover a loop invariant in order to prove the correctness of a wff. Suppose we want to prove the correctness of the following program to compute the power a^b of two natural numbers a and b, where $a > 0$ and $b \geq 0$:

```
{$(a > 0) \wedge (b \geq 0)$}
$i := 0$;
$p := 1$;
while $i < b$ do
    $p := p * a$;
    $i := i + 1$
od
{$p = a^b$}
```

The program consists of three statements. So we can represent the program and its precondition and postcondition in the following form:

$$\{(a > 0) \wedge (b \geq 0)\}\ S_1;\ S_2;\ S_3\ \{p = a^b\}.$$

In this form, S_1 and S_2 are the first two assignment statements, and S_3 represents the while statement. The composition rule (8.12) tells us that we can prove that the wff is correct if we can find proofs of the following three statements for some wffs P and Q.

$$\begin{aligned}&\{(a > 0) \wedge (b \geq 0)\}\, S_1 \,\{Q\},\\ &\{Q\}\, S_2 \,\{P\},\\ &\{P\}\, S_3 \,\{p = a^b\}.\end{aligned}$$

Where do P and Q come from? If we know P, then we can use AA to work backward through S_2 to find Q. But how do we find P? Since S_3 is a while statement, P should be a loop invariant. So we need to do a little work.

From (8.15) we know that a loop invariant P for the while statement S_3 must satisfy the following form:

$$\{P\}\ \textbf{while}\ i < b\ \textbf{do}\ p := p * a;\ i := i + 1\ \textbf{od}\ \{P \wedge \neg\,(i < b)\}.$$

Let's try some possibilities for P. Suppose we set $P \wedge \neg\,(i < b)$ equivalent to the program's postcondition $p = a^b$ and try to solve for P. This won't work because $p = a^b$ does not contain the letter i. So we need to be more flexible in our thinking. Since we have the consequence rule, all we really need is an invariant P such that $P \wedge \neg\,(i < b)$ implies $p = a^b$.

After staring at the program, we might notice that the equation $p = a^i$ holds both before and after the execution of the two assignment statements in the body of the while statement. It's also easy to see that the inequality $i \leq b$ holds before and after the execution of the body. So let's try the following definition for P:

$$(p = a^i) \wedge (i \leq b).$$

This P has more promise. Notice that $P \wedge \neg (i < b)$ implies $i = b$, which gives us the desired postcondition $p = a^b$. Next, by working backward from P through the two assignment statements, we wind up with the statement

$$(1 = a^0) \wedge (0 \leq b).$$

This statement can certainly be derived from the precondition $(a \geq 0) \wedge (b > 0)$. So P does OK from the start of the program down to the beginning of the while loop. All that remains is to prove the following statement:

$$\{P\} \textbf{ while } i < b \textbf{ do } p := p * a;\ i := i + 1 \textbf{ od } \{P \wedge \neg (i < b)\}.$$

By (8.15), all we need to prove is the following statement:

$$\{P \wedge (i < b)\}\ p := p * a;\ i := i + 1\ \{P\}.$$

This can be done easily, working backward from P through the two assignment statements. We'll put everything together in the following example.

example 8.9 Using the While Rule

We'll prove the correctness of the following program to compute the power a^b of two natural numbers a and b, where $a > 0$ and $b \geq 0$:

```
{$(a > 0) \wedge (b \geq 0)$}
$i := 0$;
$p := 1$;
while $i < b$ do
    $p := p * a$;
    $i := i + 1$
od
{$p = a^b$}
```

We'll use the loop invariant $P = (p = a^i) \wedge (i \leq b)$ for the while statement. To keep things straight, we'll insert $\{P\}$ as the precondition for the while loop and $\{P \wedge \neg (i < b)\}$ as the postcondition for the while loop as follows:

```
{$(a > 0) \wedge (b \geq 0)$}
$i := 0;$
$p := 1;$
{$P$} = {$(p = a^i) \wedge (i \leq b)$}
while $i < b$ do
    $p := p * a;$
    $i := i + 1$
od
{$P \wedge \neg C$} = {$(p = a^i) \wedge (i \leq b) \wedge \neg (i < b)$}
{$p = a^b$}
```

We'll start by proving that $P \wedge \neg C \rightarrow (p = a^b)$.

1.	$(p = a^i) \wedge (i \leq b) \wedge \neg (i < b)$	P
2.	$p = a^i$	1, Simp
3.	$(i \leq b) \wedge \neg (i < b)$	1, Simp
4.	$i = b$	3, T
5.	$p = a^b$	2, 4, EE
6.	$(p = a^i) \wedge (i \leq b) \wedge \neg (i < b) \rightarrow (p = a^b)$	1, 5, CP

Next, we'll prove the correctness of $\{P\}$ **while** $i < b$ **do** S $\{P \wedge \neg (i < b)\}$. The while inference rule tells us to prove the correctness of $\{P \wedge (i < b)\}$ S $\{P\}$.

7.	$\{(p = a^{i+1}) \wedge (i + 1 \leq b)\}$ $i := i + 1$ $\{(p = a^i) \wedge (i \leq b)\}$	AA
8.	$\{(p * a = a^{i+1}) \wedge (i + 1 \leq b)\}$ $p := p * a$ $\{(p = a^{i+1}) \wedge (i + 1 \leq b)\}$	AA
9.	$(p = a^i) \wedge (i \leq b) \wedge (i < b)$	P
10.	$p = a^i$	9, Simp
11.	$i < b$	9, Simp
12.	$b < i + 1$	P for IP
13.	$(i < b) \wedge (b < i + 1)$	11, 12, Conj
14.	false	13, T (for integers i and b)
15.	$i + 1 \leq b$	12, 14, IP
16.	$a = a$	EA
17.	$p * a = a^{i+1}$	10, 16, EE
18.	$(p * a = a^{i+1}) \wedge (i + 1 \leq b)$	15, 17, Conj
19.	$P \wedge (i < b) \rightarrow (p * a = a^{i+1}) \wedge (i + 1 \leq b)$	9, 18, CP
20.	$\{P \wedge (i < b)\}$ $p := p * a;$ $i := i + 1$ $\{P\}$	7, 8, 19, Comp, Conseq
21.	$\{P\}$ **while** $i < b$ **do** $p := p * a;$ $i := i + 1$ **od** $\{P \wedge \neg (i < b)$	20, While

Now let's work on the two assignment statements that begin the program. So we'll prove the correctness of $\{(a > 0) \wedge (b \geq 0)\}\ i := 0;\ p := 1\ \{P\}$.

22.	$\{(1 = a^i) \wedge (i \leq b)\}\ p := 1\ \{(p = a^i) \wedge (i \leq b)\}$	AA
23.	$\{(1 = a^0) \wedge (0 \leq b)\}\ i := 0\ \{(1 = a^i) \wedge (i \leq b)\}$	AA
24.	$\quad (a > 0) \wedge (b \geq 0)$	P
25.	$\quad a > 0$	24, Simp
26.	$\quad b \geq 0$	24, Simp
27.	$\quad 1 = a^0$	25, T
28.	$\quad (1 = a^0) \wedge (0 \leq b)$	26, 27, Conj
29.	$(a > 0) \wedge (b \geq 0) \rightarrow (1 = a^0) \wedge (0 \leq b)$	24, 28, CP
30.	$\{(a > 0) \wedge (b \geq 0)\}\ i := 0;\ p := 1\ \{P\}$	22, 23, 29, Comp, Conseq

The proof is finished by using the Composition and Consequence rules:

QED	30, 21, 6, Comp, Conseq.

end example

8.2.2 Array Assignment

Since arrays are fundamental structures in imperative languages, we'll modify our theory so that we can handle assignment statements like $a[i] := t$. In other words, we want to be able to construct a precondition for the following partial wff:

$$\{\ \}\ a[i] := t\ \{Q\}.$$

What do we do? We might try to work backward, as with AA, and replace all occurrences of $a[i]$ in Q by t. Let's try it and see what happens. Let $Q(a[i]/t)$ denote the wff obtained from Q by replacing all occurrences of $a[i]$ by t. We'll call the following statement the "attempted" array assignment axiom:

Attempted AAA: **(8.16)**

$$\{Q\,(a\,[i]\,/t)\}\,a\,[i] := t\,\{Q\}\,.$$

Since we're calling (8.16) the Attempted AAA, let's see whether we can find something wrong with it. For example, suppose we have the following wff, where the letter i is a variable:

$$\{\text{true}\}\ a[i] := 4\ \{a[i] = 4\}.$$

This wff is clearly correct, and we can prove it with (8.16).

1.	$\{4 = 4\}\ a[i] := 4\ \{a[i] = 4\}$	Attempted AAA
2.	true $\rightarrow (4 = 4)$	T
	QED	1, 2, Consequence.

At this point, things seem OK. But let's try another example. Suppose we have the following wff, where i and j are variables:

$$\{(i = j) \wedge (a[i] = 3)\}\ a[i] := 4\ \{a[j] = 4\}.$$

This wff is also clearly correct because $a[i]$ and $a[j]$ both represent the same indexed array variable. Let's try to prove that the wff is correct by using (8.16). The first line of the proof looks like

1. $\{a[j] = 4\}\ a[i] := 4\ \{a[j] = 4\}$ Attempted AAA

Since the precondition on line 1 is not the precondition of the wff, we need to use the consequence rule, which states that we must prove the following wff:

$$(i = j) \wedge (a[i] = 3) \rightarrow (a[j] = 4).$$

But this wff is invalid because a single array element can't have two distinct values.

So we now have an example of an array assignment statement that we "know" is correct, but we don't have the proper tools to prove that the following wff is correct:

$$\{(i = j) \wedge (a[i] = 3)\}\ a[i] := 4\ \{a[j] = 4\}.$$

What went wrong? Well, since the expression $a[i]$ does not appear in the postcondition $\{a[j] = 4\}$, the attempted AAA (8.16) just gives us back the postcondition as the precondition. This stops us in our tracks because we are now forced to prove an invalid conditional wff.

The problem is that (8.16) does not address the possibility that i and j might be equal. So we need a more sophisticated assignment axiom for arrays. Let's start again and try to incorporate the preceding remarks. We want an axiom to fill in the precondition of the following partial wff:

$$\{\ \}\ a[i] := t\ \{Q\}.$$

Of course, we need to replace all occurrences of $a[i]$ in Q by t. But we also need to replace all occurrences of $a[j]$ in Q, where j is any arithmetic expression, by an expression that allows the possibility that $j = i$. We can do this by replacing each occurrence of $a[j]$ in Q by the following if-then-else statement:

"if $j = i$ then t else $a[j]$."

For example, if the equation $a[j] = s$ occurs in Q, then the precondition will contain the following equation:

$$(\text{if } j = i \text{ then } t \text{ else } a[j]) = s.$$

When an equation contains an if-then-else statement, we can write it without if-then-else as a conjunction of two wffs. For example, the following two statements are equivalent for terms s, t, and u:

$$\begin{aligned}&(\text{if } C \text{ then } t \text{ else } u) = s,\\ &(C \to (t = s)) \wedge (\neg\, C \to (u = s)).\end{aligned}$$

So when we use the if-then-else form in a wff, we are still within the bounds of a first-order theory with equality.

For example, if $a[j] = s$ occurs in the postcondition for the array assignment $a[i] := t$, then the precondition for the assignment should replace $a[j] = s$ with either one of the following two equivalent statements:

$$\begin{aligned}&(\text{if } j = i \text{ then } t \text{ else } a\,[j]\,) = s,\\ &((j = i) \to (t = s)) \wedge ((j \neq i) \to (a\,[j] = s)).\end{aligned}$$

Now let's put things together and state the correct axiom for array assignment.

Array Assignment Axiom (AAA) **(8.17)**

$$\{P\}\ a[i] := t\ \{Q\},$$

where P is constructed from Q by the following rules:

1. Replace all occurrences of $a[i]$ in Q by t.
2. Replace all occurrences of $a[j]$ in Q by

$$\text{“if } j = i \text{ then } t \text{ else } a[j]\text{”}.$$

Note: i and j may be any arithmetic expressions that do not contain a.

It is very important that the index expressions i and j don't contain the array name. For example, $a[a[k]]$ is *not* OK, but $a[k + 1]$ is OK. To see why we can't use arrays within arrays when applying AAA, consider the following wff:

$$\{(a[1] = 2) \wedge (a[2] = 2)\}\ a[a[2]] := 1\ \{a[a[2]] = 1\}.$$

This wff is false because the assignment statement sets $a[2] = 1$, which makes the postcondition into the equation $a[1] = 1$, contradicting the fact that $a[1] = 2$. But we can use AAA to improperly "prove" that the wff is correct, as the following sequence shows.

1.	$\{1 = 1\}\ a[a[2]] := 1\ \{a[a[2]] = 1\}$	AAA attempt with $a[a[\ldots]]$
2.	$(a[1] = 2) \wedge (a[2] = 2) \to (1 = 1)$	T
	Not QED	1, 2, Conseq.

The exclusion of arrays within arrays is not a real handicap because an assignment statement like $a[a[i]] := t$ can be rewritten as the following sequence of two assignment statements:

$$j := a[i];\ a[j] := t.$$

Similarly, a logical statement like $a[a[i]] = t$ appearing in a precondition or postcondition can be rewritten as

$$\exists x\ ((x = a[i]) \wedge (a[x] = t)).$$

Now let's see whether we can use (8.17) to prove the correctness of the wff that we could not prove before.

example

8.10 Using the Array Assignment Axiom

We want to prove the correctness of the following wff:

$$\{(i = j) \wedge (a[i] = 3)\}\ a[i] := 4\ \{a[j] = 4\}.$$

This wff represents a simple reassignment of an array element, where the index of the array element is represented by two variable names. We'll include all the details of the consequence part of the proof, which uses the conjunction form of an if-then-else equation.

Proof:	1.	$\{(\text{if } j = i \text{ then } 4 \text{ else } a[j]) = 4\}$ $a[i] := 4\ \{a[j] = 4\}$	AAA
	2.	$(i = j) \wedge (a[i] = 3)$	P
	3.	$i = j$	2, Simp
	4.	$j = i$	3, Symmetry
	5.	$4 = 4$	EA
	6.	$(j = i) \to (4 = 4)$	5, T (trivial)
	7.	$(j \neq i) \to (a[j] = 4)$	4, T (vacuous)
	8.	$((j = i) \to (4 = 4)) \wedge ((j \neq i) \to (a[j] = 4))$	6, 7, Conj
	9.	$(\text{if } j = i \text{ then } 4 \text{ else } a[j]) = 4$	8, T
	10.	$(i = j) \wedge (a[i] = 3) \to ((\text{if } j = i \text{ then } 4$ $\text{else } a[j]) = 4)$	2, 9, CP
		QED	1, 10, Conseq.

end example

8.2.3 Termination

Program correctness as we have been discussing it does not consider whether loops terminate. In other words, the correctness of the wff $\{P\}\ S\ \{Q\}$ includes the assumption that S halts. That's why this kind of correctness is called *partial correctness.* For *total correctness* we can't assume that loops terminate. We must prove that they terminate.

Introductory Example

For example, suppose we're presented with the following while loop, and the only information we know is that the variables take integer values:

```
while $x \neq y$ do                (8.18)
    $x := x - 1$;
    $y := y + 1$;
od
```

We don't have enough information to be able to tell for certain whether the loop terminates. For example, if we initialize $x = 4$ and $y = 5$, then the loop will run forever. In fact, the loop will run forever whenever $x < y$. If we initialize $x = 6$ and $y = 3$, the loop will also run forever. After a little study and thought, we can see that the loop will terminate if initially we have $x \geq y$ and $x - y$ is an even number.

This example shows that the precondition (i.e., the loop invariant) must contain enough information to decide whether the loop terminates. We're going to discuss a general method for proving termination of a loop. But first we need to discuss a few preliminary ideas.

The State of a Computation

The *state* of a computation at some point is a tuple that represents the values of the variables at that point in the computation. For example, the tuple (x, y) denotes an arbitrary state of program (8.18). For our purposes the only time a state will change is when an assignment statement is executed.

For example, let the initial state of a computation for (8.18) be (10, 6). For this state the loop condition is true because $10 \neq 6$. After the execution of the first assignment statement, the state becomes (9, 6). Then after the execution of the second assignment statement, the state becomes (9, 7). So the state changes from (10, 6) to (9, 7) after one iteration of the loop. For this state the loop condition is true because $9 \neq 7$. So a second iteration of the loop can begin. We can see that the state changes from (9, 7) to (8, 8) after the second iteration of the loop. For this state the loop condition is $8 \neq 8$, which is false, so the loop terminates.

The Termination Condition

Program (8.18) terminates for the initial state (10, 6) because with each iteration of the loop the value $x - y$ gets smaller, eventually equaling zero. In other words, $x - y$ takes on the sequence of values 4, 2, 0. This is the key point in showing loop termination. There must be some decreasing sequence of numbers that stops at some point. In more general terms, the numbers must form a decreasing sequence in some well-founded set. For example, in (8.18) the well-founded set is the set $\mathbb{N}$ of natural numbers.

To show loop termination, we need to find a well-founded set $\langle W, \prec \rangle$ together with a way to associate the state of the ith iteration of the loop with an element $x_i \in W$ such that the elements form a decreasing sequence

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

Since W is well-founded, the sequence must stop. Thus the loop must halt.

Let's put things together and describe the general process to prove termination of a program **while** C **do** S with respect to a loop invariant P: We'll assume that we already know, or we have already proven, that the body S terminates. This reflects the normal process of working from the inside out when doing termination proofs. Here's the termination condition.

Termination Condition (8.19)

The program **while** C **do** S terminates with respect to the loop invariant P if the following conditions are met, where s is the program state before the execution of S and t is the program state after the execution of S.

1. Find a well-founded set $\langle W, \prec \rangle$.
2. Find an expression f in terms of the program variables.
3. Prove that if P and C are true for state s, then

$$f(s), f(t) \in W \text{ and } f(s) \succ f(t).$$

Proof: Notice that (8.19) requires that $f(s)$, $f(t) \in W$. The reason for this is that f is an expression that may very well not even be defined for certain states or it may be defined but not a member of W. So we must check that $f(s)$ and $f(t)$ are defined and members of W are in good standing. Then the statement $f(s) \succ f(t)$ will ensure that the loop terminates. So assume we have met all the conditions of (8.19). Let s_i represent the state prior to the ith execution of S. Then s_{i+1} represents the state after the ith execution of S. Therefore, we have the following decreasing sequence of elements in W:

$$f(s_1) \succ f(s_2) \succ f(s_3) \succ \cdots .$$

Since W is a well-founded set, this sequence must stop because all descending chains must be finite. Therefore the loop terminates. QED.

example 8.11 A Termination Proof

Let's show that the following program terminates with respect to the loop invariant $P = (x \geq y) \wedge \text{even}(x - y)$ where all variables take integer values:

```
while $x \neq y$ do
    $x := x - 1$;
    $y := y + 1$;
od
```

We'll leave the correctness proof with respect to P as an exercise. For a well-founded set we'll choose $\mathbb{N}$ with the usual ordering, and for the program variables (x, y) we'll define

$$f(x, y) = x - y.$$

If $s = (x, y)$ is the state before the execution of the loop's body and t is the state after execution of the loop's body, then

$$t = (x - 1, y + 1).$$

So the expressions $f(s)$ and $f(t)$ become

$$f(s) = f(x, y) = x - y,$$
$$f(t) = f(x - 1, y + 1) = (x - 1) - (y + 1) = x - y - 2.$$

To prove that program terminates with respect to P, we must prove the following statement.

If P and C are true for state s, then $f(s), f(t) \in \mathbb{N}$ and $f(s) > f(t)$.

So assume that P and C are true for state s. This means that the following two statements are true.

$$(x \geq y) \wedge \text{even}(x - y) \quad \text{and} \quad (x \neq y).$$

Since $x \geq y$ and $x \neq y$, it follows that $x > y$, so $x - y > 0$. Therefore, $f(s) \in \mathbb{N}$. Since $x - y$ is even and positive, it follows that $x - y - 2 \geq 0$. So $f(t) \in \mathbb{N}$. Finally, since $x - y > x - y - 2$, it follows that $f(s) > f(t)$. Therefore, the program terminates with respect to P.

end example

example 8.12 A Termination Proof

Let's look at a popular example of termination that needs a well-founded set other than the natural numbers. Suppose we have the following while loop where integer(x) means x is an integer and random( ) is a random number generator that returns a natural number.

```
{integer (x) }
while x ≠ 0 do
   if x < 0 then x := random( ) else x := x − 1 fi
od
{integer (x) ∧ x = 0}
```

After some study it becomes clear that the program terminates because if x is initially a negative integer, then it is assigned a random natural number. So after at most one iteration of the loop, x is a natural number. Subsequent iterations decrement x to zero, which terminates the loop.

To prove termination from a formal point of view we need to find a well-founded set W and an expresion f such that $f(s) \succ f(t)$ where s represents x at the beginning the loop body ($s = x$) and t represents x at the end of the loop body (either t is a random natural number or $t = x - 1$). Since we don't know in advance whether x is negative, we don't know how many times the loop will execute because it depends on the random natural number that is generated. So we can't define $W = \mathbb{N}$ and $f(x) = x$ because $f(x)$ may not be in $\mathbb{N}$.

But we can get the job done with the well-founded set $W = \mathbb{N} \times \mathbb{N}$ with the lexicographic ordering. Then we can define

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

Notice, for example, that

$$f(0) \prec f(1) \prec f(2) \prec \cdots \prec f(-1) \prec f(-2) \prec f(-3) \prec \cdots .$$

Thus we have $f(s), f(t) \in W$ and $f(s) \succ f(t)$. Therefore, (8.19) tells us that the loop terminates.

end example

As a final remark to this short discussion, we should remember the fundamental requirement that programs with loops need loop invariants that contain enough restrictions to ensure that the loops terminate.

Note

Hopefully, this introduction has given you the flavor of proving properties of programs. There are many mechanical aspects to the process. For example, the backwards application of the AA and AAA rules is a simple substitution problem that can be automated. We've omitted many important results. For example, if the programming language has other control structures, such as for-loops and repeat-loops, then new inference rules must be constructed. The original papers in these areas are by Hoare [1969] and Floyd [1967]. A good place to start reading more about this subject is the survey paper by Apt [1981].

Different languages usually require different formal theories to handle the program correctness problem. For example, declarative languages, in which programs can consist of recursive definitions, require methods of inductive proof in their formal theories for proving program correctness.

Exercises

Assignment Statements

1. Prove that the following wff is correct over the domain of integers:

$$\{\text{true} \wedge \text{even}(x)\}\ y := x + 1\ \{\text{odd}(y)\}.$$

2. Prove that each of the following wffs is correct. Assume that the domain is the set of integers.

   a. $\{(a > 0) \wedge (b > 0)\}\ x := a;\ y := b\ \{x + y > 0\}$.
   b. $\{a > b\}\ x := -a;\ y := -b\ \{x < y\}$.

3. Both of the following wffs claim to correctly perform the swapping process. The first one uses a temporary variable. The second does not. Prove that each wff is correct. Assume that the domain is the real numbers.

   a. $\{x < y\}$ temp $:= x;\ x := y;\ y :=$ temp $\{y < x\}$.
   b. $\{x < y\}\ y := y + x;\ x := y - x;\ y := y - x\ \{y < x\}$.

If-Then and If-Then-Else Statements

4. Prove that each of the following wffs is correct. Assume that the domain is the set of integers.

   a. $\{x < 10\}$ **if** $x \geq 5$ **then** $x := 4\ \{x < 5\}$.
   b. $\{\text{true}\}$ **if** $x \neq y$ **then** $x := y\ \{x = y\}$.
   c. $\{\text{true}\}$ **if** $x < y$ **then** $x := y\ \{x \geq y\}$.
   d. $\{\text{true}\}$ **if** $x > y$ **then** $x := y + 1;\ y := x + 1$ **fi** $\{x \leq y\}$.

5. Prove that each of the following wffs is correct. Assume that the domain is the set of integers.

   a. $\{\text{true}\}$ **if** $x < y$ **then** max $:= y$ **else** max $:= x$ $\{(\text{max} \geq x) \wedge (\text{max} \geq y)\}$.
   b. $\{\text{true}\}$ **if** $x < y$ **then** $y := y - 1$ **else** $x := -x;\ y := -y$ **fi** $\{x \leq y\}$.

6. Show that each of the following wffs is *not* correct over the domain of integers.

   a. $\{x < 5\}$ **if** $x \geq 2$ **then** $x := 5\ \{x = 5\}$.
   b. $\{\text{true}\}$ **if** $x < y$ **then** $y := y - x\ \{y > 0\}$.

While Statements

7. Prove that the following wff is correct, where x and y are integers:

```
{$x \geq y \wedge$ even$(x - y)$}
while $x \neq y$ do
    $x := x - 1$;
    $y := y + 1$;
od
{$x \geq y \wedge$ even$(x - y) \wedge (x = y)$}.
```

8. Prove that each of the following wffs is correct.

   a. The program computes the floor of a nonnegative real number x. *Hint:* Let the loop invariant be $(i \leq x)$.

```
{$x \geq 0$}
$i := 0$;
while $i \leq x - 1$ do $i := i + 1$ od
{$i =$ floor$(x)$}.
```

   b. The program computes the floor of a negative real number x. *Hint:* Let the loop invariant be $(x < i + 1)$.

```
{$x < 0$}
$i := -1$;
while $x < i$ do $i := i - 1$ od
{$i =$ floor$(x)$}.
```

   c. The program computes the floor of an arbitrary real number x, where the statements S_1 and S_2 are the two programs from parts (a) and (b).

$$\{\text{true}\}\ \textbf{if}\ x \geq 0\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \{i = \text{floor}(x)\}.$$

9. Given a natural number n, the following program computes the sum of the first n natural numbers. Prove that the wff is correct. *Hint:* Let the loop

invariant be $(s = i(i + 1)/2) \wedge (i \leq n)$.

```
$\{n \geq 0\}$
$i := 0;$
$s := 0;$
while $i < n$ do
    $i := i + 1;$
    $s := s + i$
od
$\{s = n(n+1)/2\}.$
```

10. The following program implements the division algorithm for natural numbers. It computes the quotient and the remainder of the division of a natural number by a positive natural number. Prove that the wff is correct. *Hint:* Let the loop invariant be $(a = yb + x) \wedge (0 \leq x)$.

```
$\{(a \geq 0) \wedge (b > 0)\}$
$x := a;$
$y := 0;$
while $b \leq x$ do
    $x := x - b;$
    $y := y + 1$
od;
$r := x;$
$q := y$
$\{(a = qb + r) \wedge (0 \leq r < b)\}.$
```

11. (*Greatest Common Divisor*). The following program claims to find the greatest common divisor gcd(a, b) of two positive integers a and b. Prove that the wff is correct.

```
$\{(a > 0) \wedge (b > 0)\}$
$x := a;$
$y := b;$
while $x \neq y$ do
    if $x > y$ then $x := x - y$ else $y := y - x$
od;
great $:= x$
$\{\gcd(a, b) = \text{great}\}.$
```

Hints: Use $\gcd(a, b) = \gcd(x, y)$ as the loop invariant. You may use the following useful fact derived from (2.2) for any integers w and z: $\gcd(w, z) = \gcd(w - z, z)$.

12. Write a program to compute the ceiling of an arbitrary real number. Give the program a precondition and a postcondition, and prove that the resulting wff is correct. *Hint:* Look at Exercise 8.

Array Assignment

13. For each of the following partial wffs, fill in the precondition that results by applying the array assignment axiom (8.17).

    a. $\{\ \}\ a[i - 1] := 24\ \{a[j] = 24\}$.
    b. $\{\ \}\ a[i] := 16\ \{(a[i] = 16) \wedge (a[j + 1] = 33)\}$.
    c. $\{\ \}\ a[i + 1] := 25;\ a[j - 1] := 12\ \{(a[i] = 12) \wedge (a[j] = 25)\}$.

14. Prove that each of the following wffs is correct.

    a. $\{(i = j + 1) \wedge (a[j] = 39)\}\ a[i - 1] := 24\ \{a[j] = 24\}$.
    b. $\{\text{even}(a[i]) \wedge (i = j + 1)\}\ a[j] := a[i] + 1\ \{\text{odd}(a[i - 1])\}$.
    c. $\{(i = j - 1) \wedge (a[i] = 25) \wedge (a[j] = 12)\}$
       $a[i + 1] := 25;\ a[j - 1] := 12$
       $\{(a[i] = 12) \wedge (a[j] = 25)\}$.

15. The following wffs are not correct. For each wff, apply the array assignment axiom to the postcondition and assignment statements to obtain a condition Q. Show that the precondition does not imply Q.

    a. $\{\text{even}(a[i])\}\ a[i + 1] := a[i] + 1\ \{\text{even}(a[i + 1])\}$.
    b. $\{a[2] = 2\}\ i := a[2];\ a[i] := 1\ \{(a[i] = 1) \wedge (i = a[2])\}$.
    c. $\{\forall j\ ((1 \le j \le 5) \rightarrow (a[j] = 23))\}\ i := 3;\ a[i] := 355$
       $\{\forall j\ ((1 \le j \le 5) \rightarrow (a[j] = 23))\}$.
    d. $\{(a[1] = 2) \wedge (a[2] = 2)\}\ a[a[2]] := 1\ \{\exists x\ ((x = a[2]) \wedge (a[x] = 1))\}$.

Termination

16. Prove that each of the following loops terminates with respect to the given loop invariant P, where $\text{int}(x)$ means x is an integer and $\text{even}(x)$ means x is even. *Hint:* In each case use the well-founded set $\mathbb{N}$ with the usual ordering.

    a. **while** $i < x$ **do** $i := i + 1$ **od** with $P = \text{int}(i) \wedge \text{int}(x) \wedge (i \le x)$.
    b. **while** $i < x$ **do** $x := x - 1$ **od** with $P = \text{int}(i) \wedge \text{int}(x) \wedge (i \le x)$.
    c. **while** $\text{even}(x) \wedge (x \ne 0)$ **do** $x := x/2$ **od** with $P = \text{int}(x)$.

17. Given the following while loop:

    while $x \ne y$ **do if** $x < y$ **then** $y := y - x$ **else** $x := x - y$ **od**.

Let $P = \text{pos}(x) \wedge \text{pos}(y)$ be the loop invariant, where $\text{pos}(z)$ means z is a positive integer. Prove that the loop terminates for each of the following choices of f and well-founded set W.

a. $f(x, y) = x + y$ and $W = \mathbb{N}$.
b. $f(x, y) = \max(\text{x}, \text{y})$ and $W = \mathbb{N}$.
c. $f(x, y) = (x, y)$ and $W = \mathbb{N} \times \mathbb{N}$ with the lexicographic ordering.

18. Exercise 17 demonstrates that the following loop terminates for the loop invariant $P = \text{pos}(x) \wedge \text{pos}(y)$, where $\text{pos}(z)$ means z is a positive integer:

```
while $x \neq y$ do if $x < y$ then $y := y - x$ else $x := x - y$ od.
```

Show that if we let $W = \mathbb{N}$ with the usual ordering, then each of the following choices for f cannot be used to prove termination of the loop.

a. $f(x, y) = |x - y|$.
b. $f(x, y) = \min(x, y)$.

Challenges

19. Prove the total correctness of the following program to compute a mod b, where a and b are natural numbers with $b > 0$. *Hint:* Let the loop invariant be $\exists x\ (a = xb + r) \wedge (0 \leq r)$.

```
$\{(a \geq 0) \wedge (b > 0)\}$
$r := a;$
while $r \geq b$ do $r := r - b$ od
$\{r = a \bmod b\}$.
```

20. Let member(a, L) be the test for membership of a in list L. Prove the total correctness of the following program to compute member(a, L). *Hint:* Let the loop invariant be "member(a, x) = member(a, L)".

```
$\{\text{true}\}$
$x := L;$
while $x \neq \langle\ \rangle$ and $a \neq$ head$(L)$ do $x :=$ tail$(x)$ od;
$r := (x \neq \langle\ \rangle)$
$\{r = \text{member}(a, L)\}$.
```

8.3 Higher-Order Logics

In first-order predicate calculus the only things that can be quantified are individual variables, and the only things that can be arguments for predicates are terms (i.e., constants, variables, or functional expressions with terms as arguments). If we loosen up a little and allow our wffs to quantify other things like predicates or functions, or if we allow our predicates to take arguments that are predicates or functions, then we move to a *higher-order logic.*

Is higher-order logic necessary? The purpose of this section is to convince you that the answer is yes. After some examples we'll give a general definition that will allow us to discuss nth-order logic for any natural number n.

Some Introductory Examples

We often need higher-order logic to express simple statements about the things that interest us. We'll do a few examples to demonstrate.

example 8.13 Formalizing a Statement

Let's try to formalize the following statement:

"There is a function that is larger than the log function."

This statement asserts the existence of a function. So if we want to formalize the statement, we'll need to use higher-order logic to quantify a function. We might formalize the statement as

$$\exists f\ \forall x\ (f(x) > \log x).$$

This wff is an instance of the following more general wff, where $>$ is an instance of p and log is an instance of g.

$$\exists f\ \forall x\ p(f(x),\ g(x)).$$

end example

example 8.14 Equality

Let's formalize the notion of equality. Suppose we agree to say that x and y are identical if all their properties are the same. We'll signify this by writing $x = y$. Can we express this thought in formal logic? Sure. If P is some property, then we can think of P as a predicate, and we'll agree that $P(x)$ means that x has property P. Then we can define $x = y$ as the following higher-order wff:

$$\forall P\ ((P(x) \rightarrow P(y)) \wedge (P(y) \rightarrow P(x))).$$

This wff is higher-order because the predicate P is quantified.

end example

example **8.15 Mathematical Induction**

Suppose that we want to formalize the following version of the principle of mathematical induction:

> For any predicate P, if $P(0)$ is true and if for all natural numbers n, $P(n)$ implies $P(n + 1)$, then $P(n)$ is true for all n.

We can represent the statement with the following higher-order wff:

$$\forall P\ (P(0) \wedge \forall n(P(n) \rightarrow P(n+1)) \rightarrow \forall n\ P(n)).$$

This wff is an instance of the following more general higher-order wff, where $c = 0$ and $s(n) = n + 1$:

$$\forall P\ (P(c) \wedge \forall n(P(n) \rightarrow P(s(n))) \rightarrow \forall n\ P(n)).$$

end example

Now that we have some examples, let's get down to business and discuss higher-order logic in a general setting that allows us to classify the different orders of logic.

8.3.1 Classifying Higher-Order Logics

To classify higher-order logics, we need to make an assumption about the relationship between predicates and sets.

Identifying Sets with Predicates

We'll assume that predicates are sets and that sets are predicates. Let's see why we can think of predicates and sets as the same thing. For example, if P is a predicate with one argument, we can think of P as a set in which $x \in P$ if and only if $P(x)$ is true. Similarly, if S is a set of 3-tuples, we can think of S as a predicate in which $S(x, y, z)$ is true if and only if $(x, y, z) \in S$.

The relationship between sets and predicates allows us to look at some wffs in a new light. For example, consider the following wff:

$$\forall x\ (A(x) \rightarrow B(x)).$$

In addition to the usual reading of this wff as "For every x, if $A(x)$ is true, then $B(x)$ is true," we can now read it in terms of sets by saying, "For every x, if $x \in A$, then $x \in B$." In other words, we have a wff that represents the statement "A is a subset of B."

Definition of Higher-Order Logic

The identification of predicates and sets puts us in position to define higher-order logics.

Higher-Order Logic and Higher-Order Wff

A logic is called *higher-order* if it allows sets to be quantified or if it allows sets to be elements of other sets.

A wff that quantifies a set or has a set as an argument to a predicate is called a *higher-order wff.*

For example, the following two wffs are higher-order wffs:

$\exists S\ S(x)$	The set S is quantified.
$S(x) \wedge T(S)$	The set S is an element of the set T.

Functions are Sets

Let's see how functions fit into the picture. Recall that a function can be thought of as a set of 2-tuples. For example, if $f(x) = 3x$ for all $x \in \mathbb{N}$, then we can think of f as the set

$$f = \{(x, 3x) \mid x \in \mathbb{N}\}.$$

So whenever a wff contains a quantified function name, the wff is actually quantifying a set and thus is a higher-order wff by our definition. Similarly, if a wff contains a function name as an argument to a predicate, then the wff is higher-order. For example, the following two wffs are higher-order wffs:

$\exists f\ \forall x\ p(f(x), g(x))$	The function f is a set and is quantified.
$p(f(x)) \wedge q(f)$	The function f is a set and is an element of the set q.

Since we can think of a function as a set and we are identifying sets with predicates, we can also think of a function as a predicate. For example, let f be the function

$$f = \{(x, 3x) \mid x \in \mathbb{N}\}.$$

We can think of f as a predicate with two arguments. In other words, we can write the wff $f(x, 3x)$ and let it mean "x is mapped by f to $3x$," which of course we usually write as $f(x) = 3x$.

Classifying Orders of Logic

Now let's see whether we can classify the different orders of logic. We'll start with the two logics that we know best. A propositional calculus is called a *zero-order logic* and a first-order predicate calculus is called a *first-order logic.* We want to continue the process by classifying the higher-order logics as second-order, third-order, and so on. To do this, we need to attach an order to each predicate and each quantifier that occurs in a wff. We'll start with the *order of a predicate.*

The Order of a Predicate

A predicate has *order* 1 if all its arguments are terms (i.e., constants, individual variables, or function values). Otherwise, the predicate has *order* $n + 1$, where n is the highest order among its arguments that are not terms.

For example, for each of the following wffs we've given the order of its predicates (i.e., sets):

$S(x) \wedge T(S)$	S has order 1, and T has order 2.
$p(f(x)) \wedge q(f)$	p has order 1, f has order 1, and q has order 2.

The reason that the function f has order 1 is that any function, when thought of as a predicate, takes only terms for arguments. Thus any function name has order 1. Remember to distinguish between $f(x)$ and f; $f(x)$ is a term, and f is a function (i.e., a set or a predicate).

We can also relate the order of a predicate to the level of nesting of its arguments, where we think of a predicate as a set. For example, if a wff contains the three statements $S(x)$, $T(S)$, and $P(T)$, then we have $x \in S$, $S \in T$, and $T \in P$. The orders of S, T, and P are 1, 2, and 3. So the order of a predicate (or set) is the maximum number of times the symbol $\in$ is used to get from the set down to its most basic elements.

Now we'll define the *order of a quantifier* as follows:

The Order of a Quantifier

A quantifer has *order* 1 if it quantifies an individual variable. Otherwise, the quantifier has *order* $n + 1$, where n is the order of the predicate being quantified.

For example, let's find the orders of the quantifiers in the wff that follows. Try your luck before you read the answers.

$$\forall x\ \exists S\ \exists T\ \exists f\ (S(x, f(x)) \wedge T(S)).$$

The quantifier $\forall x$ has order 1 because x is an individual variable. $\exists S$ has order 2 because S has order 1. $\exists T$ has order 3 because T has order 2. $\exists f$ has order 2 because f is a function name, and all function names have order 1.

Now we can make a simple definition for the order of a wff.

The Order of a Wff

The *order of a wff* is the highest order of any of its predicates and quantifiers.

example **8.16 Orders of Wffs**

Here are a few sample wffs and their orders.

First-Order Wffs	
$S(x)$	S has order 1.
$\forall x\ S(x)$	Both S and $\forall x$ have order 1.
Second-Order Wffs	
$S(x) \wedge T(S)$	S has order 1, and T has order 2.
$\exists S\ S(x)$	S has order 1, and $\exists S$ has order 2.
$\exists S\ (S(x) \wedge T(S))$	S has order 1, and $\exists S$ and T have order 2.
$P(x, f, f(x))$	P has order 2 because f has order 1.
Third-Order Wffs	
$S(x) \wedge T(S) \wedge P(T)$	S, T, and P have orders 1, 2, and 3.
$\forall T\ (S(x) \wedge T(S))$	S, T, $\forall T$ have orders 1, 2, and 3.
$\exists T\ (S(x) \wedge T(S) \wedge P(T))$	S, T, P, and $\exists T$ have orders 1, 2, 3, and 3.

end example

Now we can make the definition of a nth-order logic.

The Order of a Logic
An *nth-order logic* is a logic whose wffs have order n or less.

Let's do some examples that transform sentences into higher-order wffs.

example **8.17 Subsets**

Suppose we want to represent the following statement in formal logic.

"There is a set of natural numbers that doesn't contain 4."

Since the statement asserts the existence of a set, we'll need an existential quantifier. The set must be a subset of the natural numbers, and it must not contain the number 4. Putting these ideas together, we can write a mixed version (informal and formal) as follows:

$$\exists S\ (S \text{ is a subset of } \mathbb{N} \text{ and } \neg\ S(4)).$$

Let's see whether we can finish the formalization. We've seen that the general statement "A is a subset of B" can be formalized as follows:

$$\forall x\ (A(x) \rightarrow B(x)).$$

Therefore, we can write the following formal version of our statement:

$$\exists S\ (\forall x\ (S(x) \rightarrow \mathbb{N}(x)) \wedge \neg\ S(4)).$$

This wff is second-order because S has order 1, so $\exists S$ has order 2.

end example

example **8.18 Cities, Streets, and Addresses**

Suppose we think of a city as a set of streets and a street as a set of house addresses. We'll try to formalize the following statement:

> "There is a city with a street named Main, and
> there is an address 1140 on Main Street."

Suppose C is a variable representing a city and S is a variable representing a street. If x is a name, then we'll let $N(S, x)$ mean that the name of S is x. A third-order logic formalization of the sentence can be written as follows:

$$\exists C\ \exists S\ (C(S) \wedge N(S, \text{Main}) \wedge S(1140)).$$

This wff is third-order because S has order 1, so C has order 2 and $\exists C$ has order 3.

end example

8.3.2 Semantics

How do we attach a meaning to a higher-order wff? The answer is that we construct an interpretation for the wff. We start out by specifying a domain D of individuals that we use to give meaning to the constants, the free variables, and the functions and predicates that are not quantified. The quantified individual variables, functions, and predicates are allowed to vary over all possible meanings in terms of D.

Let's try to make the idea of an interpretation clear with some examples.

example **8.19 A Second-Order Interpretation**

We'll give an interpretation for the following second-order wff:

$$\exists S\ \exists T\ \forall x\ (S(x) \rightarrow \neg\ T(x)).$$

Suppose we let the domain be $D = \{a, b\}$. We observe that S and T are predicates of order 1, and they are both quantified. So S and T can vary over all possible single-argument predicates over D. For example, the following list shows the four possible predicate definitions for S together with the corresponding set definitions for S:

Predicate Definitions for S	*Set Definitions for S*
$S(a)$ and $S(b)$ are both true.	$S = \{a, b\}$.
$S(a)$ is true and $S(b)$ is false.	$S = \{a\}$.
$S(a)$ is false and $S(b)$ is true.	$S = \{b\}$.
$S(a)$ and $S(b)$ are both false.	$S = \varnothing$.

We can see from this list that there are as many possibilities for S as there are subsets of D. A similar statement holds for T. Now it's easy to see that our example wff is true for our interpretation. For example, if we choose $S = \{a, b\}$ and $T = \varnothing$, then S is always true and T is always false. Thus

$$S(a) \to \neg\ T(a) \text{ and } S(b) \to \neg\ T(b) \text{ are both true.}$$

Therefore, $\exists S\ \exists T\ \forall x\ (S(x) \to \neg\ T(x))$ is true for the interpretation.

end example

example **8.20 A Second-Order Interpretation**

We'll give an interpretation for the following second-order wff:

$$\exists S\ \forall x\ \exists y\ S(x, y).$$

Let $D = \{a, b\}$. Since S takes two arguments, it has 16 possible definitions, one corresponding to each subset of 2-tuples over D. For example, if $S = \{(a, a), (b, a)\}$, then $S(a, a)$ and $S(b, a)$ are both true, and $S(a, b)$ and $S(b, b)$ are both false. Thus the wff $\exists S\ \forall x\ \exists y\ S(x, y)$ is true for our interpretation.

end example

example **8.21 A Third-Order Interpretation**

We'll give an interpretation for the following third-order wff:

$$\exists T\ \forall x\ (T(S) \to S(x)).$$

We'll let $D = \{a, b\}$. Since S is not quantified, it is a normal predicate and we must give it a meaning. Suppose we let $S(a)$ be true and $S(b)$ be false. This is represented by $S = \{a\}$. Now T is an order 2 predicate because it takes an order 1 predicate as its argument. T is also quantified, so it is allowed to vary over all possible predicates that take arguments like S.

From the viewpoint of sets, the arguments to T can be any of the four subsets of D. Therefore, T can vary over any of the 16 subsets of $\{\varnothing, \{a\}, \{b\}, \{a, b\}\}$. For example, one possible value for T is $T = \{\varnothing, \{b\}\}$. If we think of T as a predicate, this means that $T(\varnothing)$ and $T(\{b\})$ are both true, while $T(\{a\})$ and $T(\{a, b\})$ are both false. This value of T makes the wff $\forall x\ (T(S) \to S(x))$ true. Thus the wff $\exists T\ \forall x\ (T(S) \to S(x))$ is true for our interpretation.

end example

As we have shown in the examples, we can give interpretations to higher-order wffs. This means that we can also use the following familiar terms in our discussions about higher-order wffs.

model, countermodel, valid, invalid, satisfiable, and unsatisfiable.

What about formal reasoning with higher-order wffs? That's next.

8.3.3 Higher-Order Reasoning

Gödel proved a remarkable result in 1931. He proved that if a formal system is powerful enough to describe all the arithmetic formulas of the natural numbers and the system is consistent, then it is not complete. In other words, there is a valid wff that can't be proven as a theorem in the system. Even if additional axioms were added to make the wff provable, then there would exist a new valid wff that is not provable in the larger system. A very readable account of Gödel's proof is given by Nagel and Newman [1958].

The formulas of arithmetic can be described in a first-order theory with equality, so it follows from Gödel's result that first-order theories with equality are not complete. Similarly, we can represent the idea of equality with second-order predicate calculus. So it follows that second-order predicate calculus is not complete.

What does it really mean when we have a logic that is not complete? It means that we might have to leave the formalism of the logic to prove that some wffs are valid. In other words, we may need to argue informally—using only our wits and imaginations—to prove some logical statements. In some sense this is nice because it justifies our existence as reasoning beings. Since most theories cannot be captured by using only first-order logic, there will always be enough creative work for us to do—perhaps aided by computers.

Even though higher-order logic does not give us completeness, we can still do formal reasoning to prove the validity of many higher-order wffs.

Formal Proofs

Before we give some formal proofs, we need to say something about inference rules for quantifiers in higher-order logic. We'll use the same rules that we used for first-order logic, but we'll apply them to higher-order wffs when the need arises. Basically, the rules are a reflection of our natural discourse. To keep things clear, when we use an uppercase letter like P as a variable, we'll sometimes use the lowercase letter p to represent an instantiation of P. Here is an example to demonstrate the ideas.

example 8.22 A Valid Second-Order Wff

We'll give an informal proof and a formal proof that the following second-order wff is valid:

$$\exists P \, \forall Q \, \forall x \, (Q(x) \to P(x)).$$

Proof: Let I be an interpretation with domain D. Then the wff has the following meaning with respect to I: There is a subset P of D such that for every subset Q of D it follows that $x \in Q$ implies $x \in P$. This statement is true because we can choose P to be D. So I is a model for the wff. Since I was an arbitrary interpretation, it follows that the wff is valid. QED.

In the formal proof, we'll use EI to instantiate Q to a particular value represented by lowercase q. Also, since P is universally quantified, we can use UI to instantiate P to any predicate. We'll instantiate P to true, so that $P(x)$ becomes true(x), which means that true(x) = true.

Proof:			
	1.	$\neg \exists P \, \forall Q \, \forall x \, (Q(x) \to P(x))$	P for IP
	2.	$\forall P \, \exists Q \, \exists x \, (Q(x) \wedge \neg P(x))$	1, T
	3.	$\exists Q \, \exists x \, (Q(x) \wedge \neg \text{true}(x))$	2, UI (instantiate P to true)
	4.	$\exists x \, (q(x) \wedge \neg \text{true}(x))$	3, EI
	5.	$q(c) \wedge \neg \text{true}(c)$	4, EI
	6.	$q(c) \wedge \text{false}$	5, T
	7.	false	6, T
		QED	1, 6, IP.

end example

We'll finish the section by making a short visit with geometry.

A Euclidean Geometry Example

Let's see how higher-order logic comes into play when we discuss elementary geometry. From an informal viewpoint, the wffs of Euclidean geometry are English sentences. For example, the following four statements describe part of Hilbert's axioms for Euclidean plane geometry.

1. On any two distinct points there is always a line.

2. On any two distinct points there is not more than one line.

3. Every line has at least two distinct points.

4. There are at least three points not on the same line.

Can we formalize these axioms? Let's assume that a line is a set of points. So two lines are equal if they have the same set of points. We'll also assume that arbitrary points are denoted by the variables x, y, and z and indivdual points by the constants a, b, and c. We'll denote arbitrary lines by the variables L, M, and N and we'll denote individual lines by the constants l, m, and n. We'll let the predicate $L(x)$ denote the fact that x is a point on line L or, equivalently, L is a line on the point x. Now we can write the four axioms as second-order wffs as follows:

1. $\forall x\ \forall y\ ((x \neq y) \rightarrow \exists L\ (L(x) \wedge L(y)))$.

2. $\forall x\ \forall y\ ((x \neq y) \rightarrow \forall L\ \forall M\ (L(x) \wedge L(y) \wedge M(x) \wedge M(y) \rightarrow (L = M)))$.

3. $\forall L\ \exists x\ \exists y\ ((x \neq y) \wedge L(x) \wedge L(y))$.

4. $\exists x\ \exists y\ \exists z\ ((x \neq y) \wedge (x \neq z) \wedge (y \neq z) \wedge \forall L\ (L(x) \wedge L(y) \rightarrow \neg\ L(z)))$.

In the following examples, we'll prove that there are at least two distinct lines. In the first example, we'll give an informal proof of the statement. In the second example, we'll formalize the statement and give a formal proof.

example 8.23 An Informal Theorem and Proof

Let's prove the following theorem.

There are at least two distinct lines.

Proof: Axiom 4 tells us that there are three distinct points a, b, and c not on the same line. By Axiom 1 there is a line l on a and b, and again by Axiom 1 there is a line m on a and c. By Axiom 4, c is not on line l. Therefore, it follows that $l \neq m$. QED.

end example

example 8.24 A Formal Theorem and Proof

Now we'll formalize the theorem from Example 8.23 and give a formal proof. A formalized version of the theorem can be written as

$$\exists L\ \exists M\ \exists x\ (\neg\ L(x) \wedge M(x)).$$

Proof:

1.	$\exists x\ \exists y\ \exists z\ ((x \neq y) \wedge (x \neq z) \wedge (y \neq z)$ $\wedge\ \forall L\ (L(x) \wedge L(y) \rightarrow \neg\ L(z)))$	Axiom 4
2.	$(a \neq b) \wedge (a \neq c) \wedge (b \neq c) \wedge \forall L\ (L(a) \wedge L(b) \rightarrow \neg\ L(c))$	1,EI, EI, EI
3.	$\forall x\ \forall y\ ((x \neq y) \rightarrow \exists L\ (L(x) \wedge L(y)))$	Axiom 1
4.	$(a \neq b) \rightarrow \exists L\ (L(a) \wedge L(b))$	3, UI, UI
5.	$a \neq b$	2, Simp
6.	$\exists L\ (L(a) \wedge L(b))$	4, 5, MP
7.	$l(a) \wedge l(b)$	6, EI
8.	$(a \neq c) \rightarrow \exists L\ (L(a) \wedge L(c))$	3, UI, UI
9.	$a \neq c$	2, Simp
10.	$\exists L\ (L(a) \wedge L(c))$	8, 9, MP
11.	$m(a) \wedge m(c)$	10, EI
12.	$\forall L\ (L(a) \wedge L(b) \rightarrow \neg\ L(c))$	2, Simp
13.	$l(a) \wedge l(b) \rightarrow \neg\ l(c)$	12, UI
14.	$\neg\ l(c)$	7, 13, MP
15.	$m(c)$	11, Simp
16.	$\neg\ l(c) \wedge m(c)$	14, 15, Conj
17.	$\exists x\ (\neg\ l(x) \wedge m(x))$	16, EG
18.	$\exists M\ \exists x\ (\neg\ l(x) \wedge M(x))$	17, EG
19.	$\exists L\ \exists M\ \exists x\ (\neg\ L(x) \wedge M(x))$.	18, EG
		QED.

end example

A few more Euclidean geometry problems (both informal and formal) are included in the exercises.

Exercises

Orders of Logic

1. State the minimal order of logic needed to describe each of the following wffs.

   a. $\forall x\ (Q(x) \rightarrow P(Q))$.

   b. $\exists x\ \forall g\ \exists p\ (q(c, g(x)) \wedge p(g(x)))$.

   c. $A(B) \wedge B(C) \wedge C(D) \wedge D(E) \wedge E(F)$.

   d. $\exists P\ (A(B) \wedge B(C) \wedge C(D) \wedge P(A))$.

e. $S(x) \wedge T(S, x) \rightarrow U(T, S, x)$.
f. $\forall x\ (S(x) \wedge T(S, x) \rightarrow U(T, S, x))$.
g. $\forall x\ \exists S\ (S(x) \wedge T(S, x) \rightarrow U(T, S, x))$.
h. $\forall x\ \exists S\ \exists T\ (S(x) \wedge T(S, x) \rightarrow U(T, S, x))$.
i. $\forall x\ \exists S\ \exists T\ \exists U\ (S(x) \wedge T(S, x) \rightarrow U(T, S, x))$.

Formalizing English Sentences

2. Formalize each of the following sentences as a wff in second-order logic.
   a. There are sets A and B such that $A \cap B = \varnothing$.
   b. There is a set S with two subsets A and B such that $S = A \cup B$.

3. Formalize each of the following sentences as a wff in an appropriate higher-order logic. Also figure out the order of the logic that you use in each case.
   a. Every state has a city named Springfield.
   b. There is a nation with a state that has a county named Washington.
   c. A house has a room with a bookshelf containing a book by Thoreau.
   d. There is a continent with a nation containing a state with a county named Lincoln, which contains a city named Central City that has a street named Broadway.
   e. Some set has a partition consisting of two subsets.

4. Formalize the basis of mathematical induction: If S is a subset of $\mathbb{N}$ and $0 \in S$ and $x \in \mathrm{S}$ implies $\mathrm{succ}(x) \in S$, then $S = \mathbb{N}$.

5. If R is a relation, let $B(R)$ mean that R is a binary relation. Formalize the following statement about relations: Every binary relation that is irreflexive and transitive is antisymmetric.

Validity

6. Show that the following wff is satisfiable and invalid:

$$\exists x\ \exists y\ (p(x, y) \rightarrow \forall Q\ (\mathrm{Q}(x) \rightarrow \neg\ \mathrm{Q}(y))).$$

7. Show that each of the following wffs is valid with an informal validity argument.
   a. $\forall S\ \exists x\ S(x) \rightarrow \exists x\ \forall S\ S(x)$.
   b. $\forall x\ \exists S\ S(x) \rightarrow \exists S\ \forall x\ S(x)$.
   c. $\exists S\ \forall x\ S(x) \rightarrow \forall x\ \exists S\ S(x)$.
   d. $\exists x\ \forall S\ S(x) \rightarrow \forall S\ \exists x\ S(x)$.

8. Give an informal proof and a formal proof that the following second-order wff is valid:

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

More Geometry

9. Use the facts from the Euclidean geometry example to give an informal proof for each of the following statements. You may use any of these statements to prove a subsequent statement.
   a. For each line there is a point not on the line.
   b. Two lines cannot intersect in more than one point.
   c. Through each point there exist at least two lines.
   d. Not all lines pass through the same point.

10. Formalize each of the following statements as a wff in second-order logic, using the variable names from the Euclidean geometry example. Then provide a formal proof for each wff.
    a. Not all points lie on the same line.
    b. Two lines cannot intersect in more than one point.
    c. Through each point there exist at least two lines.
    d. Not all lines pass through the same point.

8.4 Chapter Summary

A first-order theory is a formal treatment of some subject that uses first-order predicate calculus. We often need the idea of equality when applying logic in a formal manner to a particular subject. Equality can be added to first-order logic in such a way that the following familiar notion is included: Equals can replace—be substituted for—equals.

We can prove elementary statements about imperative programs within a first-order theory where each program is bounded by two conditions—a precondition and a postcondition. The theory uses only one axiom—the assignment axiom. Some useful inference rules are the consequence rule and the rules for composition, if-then, if-then-else, and while statements. The theory can be extended by adding axioms and inference rules for items that are normally found in imperative languages, such as arrays and other loop forms. We can prove *termination conditions that imply the termination of while loops.*

When formalizing a subject, we often need higher forms of logic to express statements. Higher-order logic extends first-order logic by allowing objects other than variables—such as predicates and function names—to be quantified and to be arguments in predicates. We can classify the order of a logic if we make the association that a predicate is a set. Even though higher-order logics are not complete, we can still reason formally within these logics just as we do in propositional logic and first-order logic.

chapter 9

Computational Logic

Let us not dream that reason can ever be popular. Passions, emotions, may be made popular, but reason remains ever the property of the few.
—Johann Wolfgang von Goethe (1749–1832)

Can reasoning be automated? The answer is yes, for some logics. In this chapter we'll discuss how to automate the reasoning process for first-order logic. We might start by automating the "natural deduction" proof techniques that we introduced in Chapters 6, 7, and 8. A problem with this approach is that there are many inference rules that can be applied in many different ways. In this chapter we'll look at a more mechanical way to perform deduction. We'll introduce a single inference rule, called resolution, that can be applied automatically by a computer. We'll also see that the resolution rule is used for the execution of logic programs.

chapter guide

Section 9.1 introduces the resolution inference rule. To understand the rule, we'll need to discuss clauses, clausal forms, substitution, and unification. We'll see how the rule can be applied in a mechanical fashion to prove theorems.

Section 9.2 introduces logic programming and shows how resolution is applied to perform the computation of a logic program. We'll also give some elementary techniques for constructing logic programs.

9.1 Automatic Reasoning

Let's look at the mechanical side of logic. We're going to introduce an inference rule that can be applied automatically. As fate would have it, the rule must be applied while trying to prove that a wff is unsatisfiable. This is not really a problem, because we know that a wff is valid if and only if its negation is unsatisfiable. In other words, if we want to prove that the wff W is valid, then we can do so by trying to prove that $\neg W$ is unsatisfiable. For example, if we want to prove the validity of the conditional $A \rightarrow B$, then we can try to prove the unsatisfiability of its negation $A \wedge \neg B$.

The new inference rule, which is called the *resolution rule*, can be applied over and over again in an attempt to show unsatisfiability. We can't present the resolution rule yet because it can be applied only to wffs that are written in a special form, called *clausal form.* So let's get to it.

9.1.1 Clauses and Clausal Forms

We need to introduce a little terminology before we can describe a clausal form. Recall that a *literal* is either an atom or the negation of an atom. For example, $p(x)$ and $\neg q(x, b)$ are literals. To distinguish whether a literal has a negation sign, we may use the terms *positive literal* and *negative literal.* $p(x)$ is a positive literal, and $\neg q(x, b)$ is a negative literal.

A *clause* is a disjunction of zero or more literals. For example, the following wffs are clauses:

$$\begin{gathered} p(x), \\ \neg q(x,b), \\ \neg p(a) \vee p(b), \\ p(x) \vee \neg q(a,y) \vee p(a). \end{gathered}$$

The clause that is a disjunction of zero literals is called the *empty clause*, and it's denoted by the following special box symbol:

$$\Box.$$

The empty clause is assigned the value false. We'll soon see why this makes sense when we discuss resolution.

A *clausal form* is the universal closure of a conjunction of clauses. In other words, a clausal form is a prenex conjunctive normal form, in which all quantifiers are universal and there are no free variables. For ease of notation we'll often represent a clausal form by the set consisting of its clauses. For example, if $S = \{C_1, \ldots, C_n\}$, where each C_i is a clause, and if $x_1, \ldots, x_m$ are the free variables in the clauses of S, then S denotes the following clausal form:

$$\forall x_1 \cdots \forall x_m (C_1 \wedge \cdots \wedge C_n).$$

For example, the following list shows five wffs in clausal form together with their corresponding sets of clauses:

Wffs in Clausal Form	*Sets of Clauses*
$\forall x\ p(x)$	$\{p(x)\}$
$\forall x \neg q(x, b)$	$\{\neg q(x, b)\}$
$\forall x\ \forall y\ (p(x) \wedge \neg q(y, b))$	$\{p(x), \neg q(y, b)\}$
$\forall x\ \forall y\ (p(y, f(x)) \wedge (q(y) \vee \neg q(a)))$	$\{p(y, f(x)), q(y) \vee \neg q(a)\}$
$(p(a) \vee p(b)) \wedge q(a, b)$	$\{p(a) \vee p(b), q(a, b)\}$

Notice that the last clausal form does not need quantifiers because it doesn't have any variables. In other words, it's a proposition. In fact, for propositions a clausal form is just a conjunctive normal form (CNF).

When we talk about an interpretation for a set S of clauses, we mean an interpretation for the clausal form that S denotes. Thus we can use the words "valid," "invalid," "satisfiable," and "unsatisfiable" to describe S because these words have meaning for the clausal form that S denotes.

It's easy to see that some wffs are not equivalent to any clausal form. For example, let's consider the following wff:

$$\forall x\ \exists y\ p(x,\ y).$$

This wff is not a clausal form, and it isn't equivalent to any clausal form because it has an existential quantifier. Since clausal forms are the things that resolution needs to work on, it's nice to know that we can associate a clausal form with each wff in such a way that the clausal form is unsatisfiable if and only if the wff is unsatisfiable. Let's see how to find such a clausal form for each wff.

Constructing Clausal Forms

To construct a clausal form for a wff, we can start by constructing a prenex conjunctive normal form for the wff. If there are no free variables and all the quantifiers are universal, then we have a clausal form. Otherwise, we need to get rid of the free variables and the existential quantifiers and still retain enough information to be able to detect whether the original wff is unsatisfiable. Luckily, there's a way to do this. The technique is due to the mathematician Thoralf Skolem (1887–1963), and it appears in his paper [1928].

Let's introduce Skolem's idea by considering the following example wff:

$$\forall x\ \exists y\ p(x,\ y).$$

In this case the quantifier $\exists y$ is inside the scope of the quantifier $\forall x$. So it may be that y depends on x. For example, if we let $p(x,\ y)$ mean "x has a successor y," then y certainly depends on x. If we're going to remove the quantifier $\exists y$ from $\forall x\ \exists y\ p(x,\ y)$, then we'd better leave some information about the fact that y may depend on x. Skolem's idea was to use a new function symbol, say f, and replace each occurrence of y within the scope of $\exists y$ by the term $f(x)$. After performing this operation, we obtain the following wff, which is now in clausal form:

$$\forall x\ p(x,\ f(x)).$$

We can describe the general method for eliminating existential quantifiers as follows:

Skolem's Rule **(9.1)**

Let $\exists x\ W(x)$ be a wff or part of a larger wff. If $\exists x$ is not inside the scope of a universal quantifier, then pick a new constant c, and

$$\text{replace } \exists x\ W(x) \text{ by } W(c).$$

If $\exists x$ is inside the scope of universal quantifiers $\forall x_1, \ldots, \forall x_n$, then pick a new function symbol f, and

$$\text{replace } \exists x\ W(x) \text{ by } W(f(x_1, \ldots, x_n)).$$

The constants and functions introduced by the rule are called *Skolem functions.*

example **9.1 Applying Skolem's Rule**

Let's apply Skolem's rule to the following wff:

$$\exists x\ \forall y\ \forall z\ \exists u\ \forall v\ \exists w\ p(x, y, z, u, v, w).$$

Since the wff contains three existential quantifiers, we'll use (9.1) to create three Skolem functions to replace the existentially quantified variables as follows:

Replace x by b because $\exists x$ is not in the scope of a universal quantifier.

Replace u by $f(y, z)$ because $\exists u$ is in the scope of $\forall y$ and $\forall z$.

Replace w by $g(y, z, v)$ because $\exists w$ is in the scope of $\forall y$, $\forall z$, and $\forall v$.

Now we can apply (9.1) to eliminate the existential quantifiers by making the above replacements to obtain the following clausal form:

$$\forall y\ \forall z\ \forall v\ p(b, y, z, f(y, z), v, g(y, z, v)).$$

end example

Now we have the ingredients necessary to construct clausal forms with the property that a wff and its clausal form are either both unsatisfiable or both satisfiable.

Skolem's Algorithm **(9.2)**

Given a wff W, there exists a clausal form such that W and the clausal form are either both unsatisfiable or both satisfiable. In other words, W has a model if and only if the clausal form has a model. The clausal form can be constructed from W by the following steps:

1. Construct the prenex conjunctive normal form of W.
2. Replace all occurrences of each free variable by a new constant.
3. Use Skolem's rule (9.1) to eliminate the existential quantifiers.

Proof: We already know that any wff is equivalent to its prenex conjunctive normal form, and we also know that any wff W with free variables has a model if and only if the wff obtained from W by replacing each occurrence of a free variable by a new constant has a model. So we may assume that W is already in prenex conjunctive normal form with no free variables.

Let $s(W)$ denote the clausal form constructed from W by Skolem's algorithm. We must show that W has a model if and only if $s(W)$ has a model. One direction is easy because $s(W) \to W$ is valid. To see this, start with the premise $s(W)$ and use UI to remove all universal quantifiers. Then use EG and UG to restore the quantifiers of W in proper order. So if $s(W)$ has a model, then W has a model. We'll prove the converse by induction on the number n of existential quantifiers in W.

If $n = 0$, then $W = s(W)$, so any model for W is also a model for $s(W)$. Now assume that $n > 0$ and assume that, for any wff A of the same form with less than n existential quantifiers, if A has a model, then $s(A)$ has a model. Since $n > 0$, let $\exists y$ be the leftmost existential quantifier in W. There may be universal quantifiers to the left of $\exists y$ in the prenex form. So, for some natural number k, we can write W in the form

$$W = \forall x_1 \ldots \forall x_k \; \exists y \; C(x_1, \ldots, x_k, y).$$

Now apply Skolem's rule to W by replacing y by $f(x_1, \ldots, x_k)$ to obtain the wff

$$A = \forall x_1 \ldots \forall x_k \; C(x_1, \ldots, x_k, f(x_1, \ldots, x_k)).$$

Suppose that W has a model I with domain D. Then for all $d_1, \ldots, d_k \in D$ there exists $e \in D$ such that $C(d_1, \ldots, d_k, e)$ is true with respect to I. Let J be the interpretation for A obtained from I by defining $f(d_1, \ldots, d_k) = e$ whenever $C(d_1, \ldots, d_k, e)$ is true with respect to I. It follows that J is a model for A. So if W has a model, then A has a model. Now A has fewer than n existential quantifiers. Induction tells us that if A has a model, then $s(A)$ has a model. So if W has a model, then $s(A)$ has a model. But since A is obtained from W by applying Skolem's rule, it follows that $s(A) = s(W)$. So if W has a model, then $s(W)$ has a model. QED.

Before we do some examples, let's make a couple of remarks about the steps of the algorithm. Step 2 could be replaced by the statement "Take the existential closure." But then Step 3 would remove these same quantifiers by replacing each of the newly quantified variables with a new constant name. So we saved time and did it all in one step. Step 2 can be done at any time during the process. We need Step 2 because we know that a wff and its existential closure are either both unsatisfiable or both satisfiable.

Step 3 can be applied during Step 1 after all implications have been eliminated and after all negations have been pushed to the right, but before all quantifiers have been pushed to the left. Often this will reduce the number of variables in the Skolem function. Another way to simplify the Skolem function is to push all quantifiers to the right as far as possible before applying Skolem's rule.

example 9.2 Applying Skolem's Algorithm

Suppose W is the following wff:

$$W = \forall x \neg p(x) \land \forall y \, \exists z \, q(y, z).$$

First we'll apply (9.2) as stated. In other words, we calculate the prenex form of W by moving the quantifiers to the left to obtain

$$\forall x \, \forall y \, \exists z \, (\neg p(x) \land q(y, z)).$$

Then we apply Skolem's rule (9.1), which says that we replace z by $f(x, y)$ to obtain the following clausal form for W.

$$\forall x \, \forall y \, (\neg p(x) \land q(y, f(x, y))).$$

Now we'll start again with W, but we'll apply (9.1) during Step 1 after all implications have been eliminated and after all negations have been pushed to the right. There is nothing to do in this regard. So, before we move the quantifiers to the left, we'll apply (9.1). In this case the quantifier $\exists z$ is only within the scope of $\forall y$, so we replace z by $f(y)$ to obtain

$$\forall x \neg p(x) \land \forall y \, q(y, f(y)).$$

Now finish constructing the prenex form by moving the universal quantifiers to the left to obtain the following clausal form for W:

$$\forall x \, \forall y \, (\neg p(x) \land q(y, f(y))).$$

So we get a simpler clausal form for W in this case.

end example

example 9.3 A Simple Clausal Form

Suppose we have a wff with no variables (i.e., a propositional wff). For example, let W be the wff

$$(p(a) \to q) \land ((q \land s(b)) \to r).$$

To find the clausal form for W, we need only apply equivalences from propositional calculus to find a CNF as follows:

$$\begin{aligned}(p(a) \to q) \wedge ((q \wedge s(b)) \to r) &\equiv (\neg p(a) \vee q) \wedge (\neg (q \wedge s(b)) \vee r)\\ &\equiv (\neg p(a) vq) \wedge ((\neg q \vee \neg s(b)) \vee r)\\ &\equiv (\neg p(a) \vee q) \wedge (\neg q \vee \neg s(b) \vee r).\end{aligned}$$

end example

example **9.4 Finding a Clausal Form**

We'll use (9.2) to find a clausal form for the following wff.

$$\exists y\ \forall x\ (p(x) \to q(x, y)) \wedge \forall x\ \exists y\ (q(x, x) \wedge s(y) \to r(x)).$$

The first step is to find the prenex conjunctive normal form. Since there are two quantifiers with the same name, we'll do some renaming to obtain the following wff:

$$\exists y\ \forall x\ (p(x) \to q(x, y)) \wedge \forall w\ \exists z\ ((q(w, w) \wedge s(z)) \to r(w)).$$

Next, we eliminate the conditionals to obtain the following wff:

$$\exists y\ \forall x\ (\neg\, p(x) \vee q(x, y)) \wedge \forall w\ \exists z\ (\neg\, (q(w, w) \wedge s(z)) \vee r(w)).$$

Now, push negation to the right to obtain the following wff:

$$\exists y\ \forall x\ (\neg\, p(x) \vee q(x, y)) \wedge \forall w\ \exists z\ (\neg\, q(w, w) \vee \neg\, s(z) \vee r(w)).$$

Next, we'll apply Skolem's rule (9.1) to eliminate the existential quantifiers and obtain the following wff:

$$\forall x\ (\neg\, p(x) \vee q(x, a)) \wedge \forall w\ (\neg\, q(w, w) \vee \neg\, s(f(w)) \vee r(w)).$$

Lastly, we push the universal quantifiers to the left and obtain the desired clausal form:

$$\forall x\ \forall w\ ((\neg\, p(x) \vee q(x, a)) \wedge (\neg\, q(w, w) \vee \neg\, s(f(w)) \vee r(w))).$$

end example

example **9.5 Finding a Clausal Form**

We'll construct a clausal form for the following wff:

$$\forall x\ (p(x) \to \exists y\ \forall z\ ((p(w) \vee q(x, y)) \to \forall w\ r(x, w))).$$

The free variable w is also used in the quantifier $\forall w$, and the quantifier $\forall z$ is superfluous. So we'll do some renaming, and we'll remove $\forall z$ to obtain the following wff:

$$\forall x\ (p(x) \to \exists y\ ((p(w) \vee q(x, y)) \to \forall z\ r(x, z))).$$

We remove the conditionals in the usual way to obtain the following wff:

$$\forall x\ (\neg\ p(x) \vee \exists y\ (\neg\ (p(w) \vee q(x,\ y)) \vee \forall z\ r(x,\ z))).$$

Next, we move negation inward to obtain the following wff:

$$\forall x\ (\neg\ p(x) \vee \exists y\ ((\neg\ p(w) \wedge \neg\ q(x,\ y)) \vee \forall z\ r(x,\ z))).$$

Now we can apply Skolem's rule (9.1) to eliminate $\exists y$ and replace the free variable w by b to get the following wff:

$$\forall x\ (\neg\ p(x) \vee ((\neg\ p(b) \wedge \neg\ q(x,\ f(x))) \vee \forall z\ r(x,\ z))).$$

Next, we push the universal quantifier $\forall z$ to the left, obtaining the following wff:

$$\forall x\ \forall z\ (\neg\ p(x) \vee ((\neg\ p(b) \wedge \neg\ q(x,\ f(x))) \vee r(x,\ z))).$$

Lastly, we distribute $\vee$ over $\wedge$ to obtain the following clausal form:

$$\forall x\ \forall z\ ((\neg\ p(x) \vee \neg\ p(b) \vee r(x,\ z)) \wedge (\neg\ p(x) \vee \neg\ q(x,\ f(x)) \vee r(x,\ z))).$$

end example

So we can transform any wff into a wff in clausal form in which the two wffs are either both unsatisfiable or both satisfiable. Since the resolution rule tests clausal forms for unsatisfiability, we're a step closer to describing the idea of resolution. Before we introduce the general idea of resolution, we're going to pause and discuss resolution for the simple case of propositions.

9.1.2 Resolution for Propositions

It's easy to see how resolution works for propositional clauses (i.e., clauses with no variables). The resolution inference rule works something like a cancellation process. It takes two clauses and constructs a new clause from them by deleting all occurrences of a positive literal p from one clause and all occurrences of $\neg\ p$ from the other clause. For example, suppose we are given the following two propositional clauses:

$$p \vee q,$$
$$\neg\ p \vee r \vee \neg\ p.$$

We obtain a new clause by first eliminating p from the first clause and eliminating the two occurrences of $\neg\ p$ from the second clause. Then we take the disjunction of the leftover clauses to form the new clause:

$$q \vee r.$$

Let's write down the resolution rule in a more general way. Suppose we have two propositional clauses of the following forms:

$$p \vee A,$$
$$\neg\, p \vee B.$$

Let $A - p$ denote the disjunction obtained from A by deleting all occurrences of p. Similarly, let $B - \neg\, p$ denote the disjunction obtained from B by deleting all occurrences of $\neg\, p$. The resolution rule allows us to infer the propositional clause

$$(A - p) \vee (B - \neg\, p).$$

Here's the rule.

Resolution Rule for Propositions **(9.3)**

$$\frac{p \vee A,\ \neg\, p \vee B}{\therefore (A - p) \vee (B - \neg\, p)}.$$

Although the rule may look strange, it's a good rule. That is, it maps tautologies to a tautology. To see this, we can suppose that $(p \vee A) \wedge (\neg\, p \vee B)$ = true. If p is true, then the equation reduces to B = true. Since $\neg\, p$ is false, we can remove all occurrences of $\neg\, p$ from B and still have $B - \neg\, p$ = true. Therefore, $(A - p) \vee (B - \neg\, p)$ = true. We obtain the same result if p is false. So the inference rule does its job.

A proof by resolution is a refutation that uses only the resolution rule. So we can define a *resolution proof* as a sequence of clauses, ending with the empty clause, in which each clause in the sequence either is a premise or is inferred by the resolution rule from two preceding clauses in the sequence. Notice that the empty clause is obtained from (9.3) when A either is empty or contains only copies of p and when B either is empty or contains only copies of $\neg\, p$. For example, the simplest version of (9.3) can be stated as follows:

$$\frac{p, \neg\, p}{\therefore \Box}.$$

In other words, we obtain the well-known tautology $p \wedge \neg\, p \rightarrow$ false.

For example, let's prove that the following clausal form is unsatisfiable:

$$(\neg\, p \vee q) \wedge (p \vee q) \wedge (\neg\, q \vee p) \wedge (\neg\, p \vee \neg\, q).$$

In other words, we'll prove that the following set of clauses is unsatisfiable:

$$\{\neg\, p \vee q,\ p \vee q,\ \neg\, q \vee p,\ \neg\, p \vee \neg\, q\}.$$

The following resolution proof does the job:

Proof:	1.	$\neg p \vee q$	P
	2.	$p \vee q$	P
	3.	$\neg q \vee p$	P
	4.	$\neg p \vee \neg q$	P
	5.	$q \vee q$	1, 2, Resolution
	6.	p	3, 5, Resolution
	7.	$\neg p$	4, 5, Resolution
	8.	$\Box$	6, 7, Resolution.
		QED	

Now let's get back on our original track, which is to describe the resolution rule for clauses of the first-order predicate calculus.

9.1.3 Substitution and Unification

When we discuss the resolution inference rule for clauses that contain variables, we'll see that a certain kind of matching is required. For example, suppose we are given the following two clauses:

$$p(x, y) \vee q(y),$$
$$r(z) \vee \neg q(b).$$

The matching that we will discuss allows us to replace all occurrences of the variable y by the constant b, thus obtaining the following two clauses:

$$p(x, b) \vee q(b),$$
$$r(z) \vee \neg q(b).$$

Notice that one clause contains $q(b)$ and the other contains its negation $\neg q(b)$. Resolution will allow us to cancel them and construct the disjunction of the remaining parts, which is the clause $p(x, b) \vee r(z)$.

We need to spend a little time to discuss the process of replacing variables by terms. If x is a variable and t is a term, then the expression x/t is called a *binding* of x to t and can be read as "x gets t" or "x is bound to t" or "x has value t" or "x is replaced by t." For example, three typical bindings are written as follows:

$$x/a, \quad y/z, \quad w/f(b, v).$$

Definition of Substitution

A *substitution* is a finite set of bindings $\{x_1/t_1, \ldots, x_n/t_n\}$, where variables $x_1, \ldots, x_n$ are all distinct and $x_i \neq t_i$ for each i. We use lowercase Greek letters to denote substitutions. The *empty substitution*, which is just the empty set, is denoted by the Greek letter ϵ.

What do we do with substitutions? We apply them to expressions, an *expression* being a finite string of symbols. Let E be an expression, and let θ be the following substitution:

$$\theta = \{x_1/t_1, \ldots, x_n/t_n\}.$$

Then the *instance* of E by θ, denoted $E\theta$, is the expression obtained from E by simultaneously replacing all occurrences of the variables $x_1, \ldots, x_n$ in E by the terms $t_1, \ldots, t_n$, respectively. We say that $E\theta$ is obtained from E by *applying* the substitution θ to the expression E. For example, if $E = p(x, y, f(x))$ and $\theta = \{x/a, y/f(b)\}$, then $E\theta$ has the following form:

$$E\theta = p(x, y, f(x))\{x/a, y/f(b)\} = p(a, f(b), f(a)).$$

If S is a set of expressions, then the *instance* of S by θ, denoted $S\theta$, is the set of all instances of expressions in S by θ. For example, if $S = \{p(x, y), q(a, y)\}$ and $\theta = \{x/a, y/f(b)\}$, then $S\theta$ has the following form:

$$S\theta = \{p(x, y), q(a, y)\}\{x/a, y/f(b)\} = \{p(a, f(b)), q(a, f(b)\}.$$

Now let's see how we can combine two substitutions θ and σ into a single substitution that has the same effect as applying θ and then applying σ to any expression.

Compostion of Substitutions

The *composition* of two substitutions θ and σ is the substitution denoted by $\theta\sigma$ that satisfies the following property for any expression E:

$$E(\theta\sigma) = (E\theta)\sigma.$$

Although we have described the composition in terms of how it acts on all expressions, we can compute $\theta\sigma$ without any reference to an expression as follows:

Computing the Composition **(9.4)**

Given the two substitutions $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ and $\sigma = \{y_1/s_1, \ldots, y_m/s_m\}$. The composition $\theta\sigma$ is constructed as follows:

1. Apply σ to the denominators of θ to get $\{x_1/t_1\sigma, \ldots, x_n/t_n\sigma\}$.
2. Delete any bindings of the form x_i/x_i from the set on line 1.
3. Delete any y_i/s_i from σ if y_i is a variable in $\{x_1, \ldots, x_n\}$.
4. $\theta\sigma$ is the union of the sets constructed on lines 2 and 3.

The process looks complicated, but it's really quite simple. It's just a formalization of the following construction: For each distinct variable v occurring in the numerators of θ and σ, apply θ and then σ to v, obtaining the expression $(v\theta)\sigma$. The composition $\theta\sigma$ consists of all bindings $v/(v\theta)\sigma$ such that $v \neq (v\theta)\sigma$.

It's also nice to know that we can always check whether we constructed a composition correctly. Just make up an example atom containing the distinct variables in the numerators of θ and σ, say, $p(v_1, \ldots, v_k)$, and then check to make sure the following equation holds:

$$((p(v_1, \ldots, v_k)\theta)\sigma) = p(v_1, \ldots, v_k)\ (\theta\sigma).$$

example

9.6 Finding a Composition

Let $\theta = \{x/f(y),\ y/z\}$ and $\sigma = \{x/a,\ y/b,\ z/y\}$. To find the composition $\theta\sigma$, we first apply σ to the denominators of θ to form the following set:

$$\{x/f(y)\sigma,\ y/z\sigma\} = \{x/f(b),\ y/y\}.$$

Now remove the binding y/y to obtain $\{x/f(b)\}$. Next, delete the bindings x/a and y/b from σ to obtain $\{z/y\}$. Finally, compute $\theta\sigma$ as the union of these two sets $\theta\sigma = \{x/f(b),\ z/y\}$.

Let's check to see whether the answer is correct. For our example atom we'll pick $p(x,\ y,\ z)$ because x, y, and z are the distinct variables occurring in the numerators of θ and σ. We'll make the following two calculations to see whether we get the same answer.

$$((p(x,y,z)\,\theta)\,\sigma) = p(f(y),z,z)\,\sigma = p(f(b),y,y),$$
$$p(x,y,z)\,(\theta\sigma) = p(f(b),y,y).$$

end example

Three simple, but useful, properties of composition are listed next. The proofs are left as exercises.

Properties of Composition **(9.5)**

For any substitutions θ and σ and any expression E the following statements hold.

1. $E(\theta\sigma) = (E\theta)\sigma$.
2. $E\epsilon = E$.
3. $\theta\epsilon = \epsilon\theta = \theta$.

A substitution θ is called a *unifier* of a finite set S of literals if $S\theta$ is a singleton set. For example, if we let $S = \{p(x, b), p(a, y)\}$, then the substitution $\theta = \{x/a, y/b\}$ is a unifier of S because

$$S\theta = \{p(a, b)\},$$

which is a singleton set.

Some sets of literals don't have a unifier, while other sets have infinitely many unifiers. The range of possibilities can be shown by the following four simple examples.

1. $\{p(x), q(y)\}$ doesn't have a unifier.
2. $\{p(x), \neg\, p(x)\}$ doesn't have a unifier.
3. $\{p(x), p(a)\}$ has a unifier. Any unifer must contain the binding x/a and yield the singleton $\{p(a)\}$. e.g., $\{x/a\}$ and $\{x/a, y/z\}$ are unifiers of the set.
4. $\{p(x), p(y)\}$ has infinitely many unifiers that can yield different singletons. e.g., $\{x/y\}$, $\{y/x\}$, and $\{x/t, y/t\}$ for any term t are all unifers of the set.

Among the unifiers of a set there is always at least one unifier that can be used to construct every other unifier. To be specific, a unifier θ for S is called a *most general unifier* (mgu) for S if for every unifier α of S there exists a substitution σ such that $\alpha = \theta\sigma$. In other words, an mgu for S is a factor of every other unifier of S. Let's look at an example.

example 9.7 A Most General Unifier

As we have noted, the set $S = \{p(x), p(y)\}$ has infinitely many unifiers that we can describe as follows:

$$\{x/y\},\ \{y/x\},\text{ and } \{x/t,\ y/t\} \text{ for any term } t.$$

The unifier $\{x/y\}$ is an mgu for S because we can write the other unifiers in terms of $\{x/y\}$ as follows: $\{y/x\} = \{x/y\}\{y/x\}$, and $\{x/t, y/t\} = \{x/y\}\{y/t\}$ for any term t. Similarly, $\{y/x\}$ is an mgu for S.

end example

Unification Algorithms

We want to find a way to construct an mgu for any set of literals. Before we do this, we need a little terminology to describe the set of terms that cause two or more literals in a set to be distinct.

If S is a set of literals, then the *disagreement set* of S is constructed in the following way.

1. Find the longest common substring that starts at the left end of each literal of S.
2. The disagreement set of S is the set of all the terms that occur in the literals of S that are immediately to the right of the longest common substring.

For example, we'll construct the disagreement set for the following set of three literals.

$$S = \{p(x, f(x), y), p(x, y, z), p(x, f(a), b)\}.$$

The longest common substring for the literals in S is the string

$$\text{“}p(x,\text{”}$$

of length four. The terms in the literals of S that occur immediately to the right of this string are $f(x)$, y, and $f(a)$. Thus the disagreement set of S is

$$\{f(x), y, f(a)\}.$$

Now we have the tools to describe a very important algorithm by Robinson [1965]. The algorithm computes, for a set of atoms, a most general unifier, if one exists.

Unification Algorithm (Robinson) **(9.6)**

Input: A finite set S of atoms.

Output: Either a most general unifier for S or a statement that S is not unifiable.

1. Set $k = 0$ and $\theta_0 = \epsilon$, and go to Step 2.
2. Calculate $S\theta_k$. If it's a singleton set, then stop (θ_k is the mgu for S). Otherwise, let D_k be the disagreement set of $S\theta_k$, and go to Step 3.
3. If D_k contains a variable v and a term t, such that v does not occur in t, then calculate the composition $\theta_{k+1} = \theta_k\{v/t\}$, set $k := k + 1$, and go to Step 2. Otherwise, stop (S is not unifiable).

The composition $\theta_k\{v/t\}$ in Step 3 is easy to compute for two reasons. The variable v doesn't occur in t, and v will never occur in the numerator of θ_k. Therefore, the middle two steps of the composition construction (9.4) don't change anything. In other words, the composition $\theta_k\{v/t\}$ is constructed by applying $\{v/t\}$ to each denominator of θ_k and then adding the binding v/t to the result.

example 9.8 Finding a Most General Unifier

Let's try the algorithm on the set $S = \{p(x, f(y)), p(g(y), z)\}$. We'll list each step of the algorithm as we go.

1. Set $\theta_0 = \epsilon$.

2. $S\theta_0 = S\epsilon = S$ is not a singleton. $D_0 = \{x, g(y)\}$.

3. Variable x doesn't occur in term $g(y)$ of D_0.
Put $\theta_1 = \theta_0 \{x/g(y)\} = \{x/g(y)\}$.

2. $S\theta_1 = \{p(g(y), f(y)), p(g(y), z)\}$ is not a singleton. $D_1 = \{f(y), z\}$.

3. Variable z does not occur in term $f(y)$ of D_1.
Put $\theta_2 = \theta_1 \{z/f(y)\} = \{x/g(y), z/f(y)\}$.

2. $S\theta_2 = \{p(g(y), f(y))\}$ is a singleton. Therefore, the algorithm terminates with the mgu $\{x/g(y), z/f(y)\}$ for the set S.

end example

example 9.9 No Most General Unifier

Let's trace the algorithm on the set $S = \{p(x), p(g(x))\}$. We'll list each step of the algorithm as we go:

1. Set $\theta_0 = \epsilon$.

2. $S\theta_0 = S\epsilon = S$, which is not a singleton. $D_0 = \{x, g(x)\}$.

3. The only choices for a variable and a term in D_0 are x and $g(x)$. But the variable x occurs in $g(x)$. So the algorithm stops, and S is not unifiable.

This makes sense too. For example, if we were to apply the substitution $\{x/g(x)\}$ to S, we would obtain the set $\{p(g(x)), p(g(g(x)))\}$, which in turn gives us the same disagreement set $\{x, g(x)\}$. So the process would go on forever. Notice that a change of variables makes a big difference. For example, if we change the second atom in S to $p(g(y))$, then the algorithm unifies the set $\{p(x), p(g(y))\}$, obtaining the mgu $\{x/g(y)\}$.

end example

The following alternative algorithm for unification is due to Martelli and Montanari [1982]. It can be used on pairs of atoms.

Unification Algorithm (Martelli-Montanari) **(9.7)**

Input: A singleton set $\{A = B\}$ where A and B are atoms or terms.

Output: Either a most general unifier of A and B or a statement that they are not unifiable.

Perform the following nondeterministic actions until no action can be performed or a halt with failure occurs. If there is no failure then the output is a set of equations of the form $\{x_1 = t_1, \ldots, x_n = t_n\}$ and the mgu is $\{x_1/t_1, \ldots, x_n/t_n\}$. Note: f and g represent function or predicate symbols.

	Equation	*Action*
1.	$f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$.	Replace the equation with the equations $s_1 = t_1, \ldots, s_n = t_n$.
2.	$f(s_1, \ldots, s_m) = g(t_1, \ldots, t_m)$ and either $f \neq g$ or $m \neq n$.	Halt with failure.
3.	$x = x$	Delete the equation.
4.	$t = x$ and t is not a variable.	Replace $t = x$ with $x = t$.
5.	$x = t$, x does not occur in t, and x occurs in another equation.	Apply the substitution $\{x/t\}$ to all other equations.
6.	$x = t$, t is not a variable, and x occurs in t.	Halt with failure.

example 9.10 Finding a Most General Unifier

Let's try the algorithm on the two atoms $p(x, f(x))$ and $p(y, f(b))$. We'll list each set of equations generated by the algorithm together with the reason for each step.

$\{p(x, f(x)) = p(y, f(b))\}$	Input
$\{x = y, f(x) = f(b)\}$	Equation (1)
$\{x = y, f(y) = f(b)\}$	Equation (5)
$\{x = y, y = b\}$	Equation (1)
$\{x = b, y = b\}$	Equation (5)

Therefore, the mgu is $\{x/b, y/b\}$.

end example

9.1.4 Resolution: The General Case

Now we've got the tools to discuss resolution of clauses that contain variables. Let's look at a simple example to help us see how unification comes into play. Suppose we're given the following two clauses:

$$p(x, a) \vee \neg\, q(x),$$
$$\neg\, p(b, y) \vee \neg\, q(a).$$

We want to cancel $p(x,\ a)$ from the first clause and $\neg\ p(b,\ y)$ from the second clause. But they won't cancel until we unify the two atoms $p(x,\ a)$ and $p(b,\ y)$. An mgu for these two atoms is $\{x/b,\ y/a\}$. If we apply this unifier to the original two clauses, we obtain the following two clauses:

$$p(b, a) \vee \neg\, q(b),$$
$$\neg\, p(b, a) \vee \neg\, q(a).$$

Now we can cancel $p(b,\ a)$ from the first clause and $\neg\ p(b,\ a)$ from the second clause and take the disjunction of what's left to obtain the following clause:

$$\neg\ q(b) \vee \neg\ q(a).$$

That's the way the resolution inference rule works when variables are present. Now let's give a detailed description of the rule.

The Resolution Inference Rule

The resolution inference rule takes two clauses and constructs a new clause. But the rule can be applied only to clauses that possess the following two properties.

Two Requirements for Resolution

1. The two clauses have no variables in common.
2. There are one or more atoms, $L_1, \ldots, L_k$, in one of the clauses and one or more literals, $\neg\ M_1, \ldots, \neg\ M_n$, in the other clause such that the set $\{L_1, \ldots, L_k, M_1, \ldots, M_n\}$ is unifiable.

The first property can always be satisfied by renaming variables. For example, the variable x is used in both of the following clauses:

$$q(b, x) \vee p(x), \quad \neg\, q(x, a) \vee p(y).$$

We can replace x in the second clause with a new variable z to obtain the following two clauses that satisfy the first property:

$$q(b, x) \vee p(x), \quad \neg\, q(z, a) \vee p(y).$$

Suppose we have two clauses that satisfy properties 1 and 2. Then they can be written in the following form, where C and D represent the other parts of each clause:

$$L_1 \vee \cdots \vee L_k \vee C \quad \text{and} \quad \neg M_1 \vee \cdots \vee \neg M_n \vee D.$$

Since the clauses satisfy the second property, we know that there is an mgu θ that unifies the set of atoms $\{L_1, \ldots, L_k, M_1, \ldots, M_n\}$. In other words, there is a unique atom N such that $N = L_i\theta = M_j\theta$ for any i and j. To be specific, we'll set

$$N = L_1\theta.$$

Now we're ready to do our cancelling. Let $C\theta - N$ denote the clause obtained from $C\theta$ by deleting all occurrences of the atom N. Similarly, let $D\theta - \neg N$ denote the clause obtained from $D\theta$ by deleting all occurrences of the atom $\neg N$. The clause that we construct is the disjunction of any literals that are left after the cancellation:

$$(C\theta - N) \vee (D\theta - \neg N).$$

Summing all this up, we can state the resolution inference rule as follows:

Resolution Rule (R) **(9.8)**

$$\frac{L_1 \vee \cdots \vee L_k \vee C, \ \neg M_1 \vee \cdots \vee \neg M_n \vee D}{\therefore (C\theta - N) \vee (D\theta - \neg N)}$$

The clause constructed in the denominator of (9.8) is called a *resolvant* of the two clauses in the numerator. Let's describe how to use (9.8) to find a resolvant of the two clauses.

1. Check the two clauses for distinct variables (rename if necessary).

2. Find an mgu θ for the set of atoms $\{L_1, \ldots, L_k, M_1, \ldots, M_n\}$.

3. Apply θ to both clauses C and D.

4. Set $N = L_1\theta$.

5. Remove all occurrences of N from $C\theta$.

6. Remove all occurrences of $\neg N$ from $D\theta$.

7. Form the disjunction of the clauses in Steps 5 and 6. This is the resolvant.

Let's do some examples to get the look and feel of resolution before we forget everything.

example

9.11 Resolving Two Clauses

We'll try to find a resolvant of the following two clauses:

$$q(b,x) \vee p(x) \vee q(b,a),$$
$$\neg\, q(y,a) \vee p(y).$$

We'll cancel the atom $q(b, x)$ in the first clause with the literal $\neg\, q(y, a)$ in the second clause. So we'll write the first clause in the form $L \vee C$, where L and C have the following values:

$$L = q(b,x) \quad \text{and} \quad C = p(x) \vee q(b,a).$$

The second clause can be written in the form $\neg\, M \vee D$, where M and D have the following values:

$$M = q(y,a) \quad \text{and} \quad D = p(y).$$

Now L and M, namely $q(b, x)$ and $q(y, a)$, can be unified by the mgu $\theta = \{y/b, x/a\}$. We can apply θ to either atom to obtain the common value $N = L\theta = M\theta = q(b, a)$. Now we can apply (9.8) to find the resolvant of the two clauses. First, compute the clauses $C\theta$ and $D\theta$:

$$C\theta = (p(x) \vee q(b,a))\{y/b, x/a\} = p(a) \vee q(b,a),$$
$$D\theta = p(y)\{y/b, x/a\} = p(b).$$

Next we'll remove all occurrences of $N = q(b, a)$ from $C\theta$ and remove all occurrences of $\neg\, N = \neg\, q(b, a)$ from $D\theta$:

$$C\theta - N = p(a) \vee q(b,a) - q(b,a) = p(a),$$
$$D\theta - \neg\, N = p(b) - \neg\, q(b,a) = p(b).$$

Lastly, we'll take the disjunction of the remaining clauses to obtain the desired resolvant $p(a) \vee p(b)$.

end example

example

9.12 Resolving Two Clauses

In this example we'll consider cancelling two literals from one of the clauses. Suppose we have the following two clauses.

$$p(f(x)) \vee p(y) \vee \neg\, q(x),$$
$$\neg\, p(z) \vee q(w).$$

We'll pick the disjunction $p(f(x)) \vee p(y)$ from the first clause to cancel with the literal $\neg\, p(z)$ in the second clause. So we need to unify the set of atoms $\{p(f(x)), p(y), p(z)\}$. An mgu for this set is $\theta = \{y/f(x),\ z/f(x)\}$. The common value N obtained by applying θ to any of the atoms in the set is $N = p(f(x))$. To see how the cancellation takes place, we'll apply θ to both of the original clauses to obtain the clauses

$$p(f(x)) \vee p(f(x)) \vee \neg\, q(x),$$
$$\neg\, p(f(x)) \vee q(w).$$

We'll cancel $p(f(x)) \vee p(f(x))$ from the first clause and $\neg\, p(f(x))$ from the second clause, with no other deletions possible. Thus the resolvant of the original two clauses is the disjunction of the remaining parts of the preceding two clauses: $\neg\, q(x) \vee q(w)$.

end example

What's so great about finding resolvants? Two things are great. One great thing is that the process is mechanical—it can be programmed. The other great thing is that the process preserves unsatisfiability. In other words, we have the following result.

Theorem **(9.9)**

Let G be a resolvant of the clauses E and F. Then $\{E, F\}$ is unsatisfiable if and only if $\{E, F, G\}$ is unsatisfiable.

Now we're almost in position to describe how to prove that a set of clauses is unsatisfiable. Let S be a set of clauses where—after possibly renaming some variables—distinct clauses of S have disjoint sets of variables. We define the *resolution* of S, denoted by $R(S)$, to be the set

$$R(S) = S \cup \{G \mid G \text{ is a resolvant of a pair of clauses in } S\}.$$

We can conclude from (9.9) that S is unsatisfiable if and only if $R(S)$ is unsatisfiable. Similarly, $R(S)$ is unsatisfiable if and only if $R(R(S))$ is unsatisfiable. We can continue on in this way. To simplify the notation, we'll define $R^0(S) = S$ and $R^{n+1}(S) = R(R^n(S))$ for $n > 0$. So for any n we can say that

$$S \text{ is unsatisfiable if and only if } R^n(S) \text{ is unsatisfiable.}$$

Let's look at some examples to demonstrate the calculation of the sequence of sets S, $R(S)$, $R^2(S)$, ...

example

9.13 A Refutation

Suppose we start with the following set of clauses:

$$S = \{p(x),\ \neg\, p(a)\}.$$

To compute $R(S)$, we must add to S all possible resolvants of pairs of clauses. There is only one pair of clauses in S, and the resolvant of $p(x)$ and $\neg\ p(a)$ is the empty clause. Thus $R(S)$ is the following set.

$$R(S) = \{p(x), \neg\ p(a), \Box\}.$$

Now let's compute $R(R(S))$. The only two clauses in $R(S)$ that can be resolved are $p(x)$ and $\neg\ p(a)$. Since their resolvant is already in $R(S)$, there's nothing new to add. So the process stops, and we have $R(R(S)) = R(S)$.

end example

example **9.14 A Refutation**

Consider the following set of three clauses.

$$S = \{p(x),\ q(y) \vee \neg\ p(y),\ \neg\ q(a)\}.$$

Let's compute $R(S)$. There are two pairs of clauses in S that have resolvants. The two clauses $p(x)$ and $q(y) \vee \neg\ p(y)$ resolve to $q(y)$. The clauses $q(y) \vee \neg\ p(y)$ and $\neg\ q(a)$ resolve to $\neg\ p(a)$. Thus $R(S)$ is the following set:

$$R(S) = \{p(x),\ q(y) \vee \neg\ p(y),\ \neg\ q(a),\ q(y),\ \neg\ p(\mathrm{a})\}.$$

Now let's compute $R(R(S))$. The two clauses $p(x)$ and $\neg\ p(a)$ resolve to the empty clause, and nothing new is added by resolving any other pairs from $R(S)$. Thus $R(R(S))$ is the following set:

$$R(R(S)) = \{p(x), q(y) \vee \neg\ p(y), \neg\ q(a), q(y), \neg\ p(a), \Box\}.$$

It's easy to see that we can't get anything new by resolving pairs of clauses in $R(R(S))$. Thus we have $R^3(S) = R^2(S)$.

end example

These two examples have something very important in common. In each case the set S is unsatisfiable, and the empty clause occurs in $R^n(S)$ for some n. This is no coincidence. The following result of Robinson [1965] allows us to test for the unsatisfiability of a set of clauses by looking for the empty clause in the sequence S, $R(S)$, $R^2(S)$,....

Resolution Theorem (9.10)

A finite set S of clauses is unsatisfiable if and only if $\Box \in R^n(S)$ for some $n \geq 0$.

The theorem provides us with an algorithm to prove that a wff is unsatisfiable. Let S be the set of clauses that make up the clausal form of the wff. Start by calculating all the resolvants of pairs of clauses from S. The new resolvants are added to S to form the larger set of clauses $R(S)$. If the empty clause has been calculated, then we are done. Otherwise, calculate resolvants of pairs of clauses in the set $R(S)$. Continue the process until we find a pair of clauses whose resolvant is the empty clause.

If we get to a point at which no new clauses are being created and we have not found the empty clause, then the process stops, and we conclude that the wff that we started with is satisfiable.

9.1.5 Theorem Proving with Resolution

Recall that a resolution proof is a sequence of clauses that ends with the empty clause, in which each clause either is a premise or can be inferred from two preceding clauses by the resolution rule. Recall also that a resolution proof is a proof of unsatisfiability. Since we normally want to prove that some wff is valid, we must first take the negation of the wff, then find a clausal form, and then attempt to do a resolution proof. We'll summarize the steps.

Steps to Prove that *W* is Valid

1. Form the negation $\neg W$. For example, if W is a conditional of the form $A \wedge B \wedge C \rightarrow D$, then $\neg W$ has the form $A \wedge B \wedge C \wedge \neg D$.
2. Use Skolem's algorithm (9.2) to convert line 1 into clausal form.
3. Take the clauses from line 2 as premises in the proof.
4. Apply the resolution rule (9.8) to derive the empty clause.

example **9.15 Binary Relations**

We'll prove that if a binary relation is irreflexive and transitive, then it is antisymmetric. If p denotes a binary relation, then the three properties can be represented as follows.

Irreflexive: $\forall x \neg p(x, x)$.

Transitive: $\forall x \forall y \forall z\ (p(x, y) \wedge p(y, z) \rightarrow p(x, z))$.

Antisymmetric: $\forall x \forall y\ (p(x, y) \rightarrow \neg p(y, x))$.

So we must prove that the wff W is valid, where

$$W = \text{Irreflexive} \wedge \text{Transitive} \rightarrow \text{Antisymmetric}.$$

To use resolution, we must prove that $\neg W$ is unsatisfiable, where

$$\neg W = \text{Irreflexive} \wedge \text{Transitive} \wedge \neg \text{Antisymmetric}.$$

Notice that $\neg$ Antisymmetric has the following form:

$\neg$ Antisymmetric $= \neg \forall x\ \forall y\ (p(x, y) \rightarrow \neg\, p(y, x)) \equiv \exists x\ \exists y\ (p(x, y) \wedge p(y, x))$.

First we put $\neg\, W$ into clausal form. The following table shows the clauses in the clausal forms for Irreflexive, Transitive, and $\neg$ Antisymmetric.

Wff	*Clauses*
$\forall x \neg\, p(x,x)$	$\neg\, p(x,x)$
$\forall x\ \forall y\ \forall z\ (p(x,y) \wedge p(y,z) \rightarrow p(x,z))$	$\neg\, p(u,v) \vee \neg\, p(v,w) \vee p(u,w)$
$\exists x\ \exists y\ (p(x,y) \wedge p(y,x))$	$p(a,b)$ and $p(b,a)$

To do a resolution proof, we start with the four clauses as premises. Our goal is to construct resolvants to obtain the empty clause. Each resolution step includes the most general unifier used for that application of resolution.

Proof:	1.	$\neg\, p(x, x)$	P
	2.	$\neg\, p(u, v) \vee \neg\, p(v, w) \vee p(u, w)$	P
	3.	$p(a, b)$	P
	4.	$p(b, a)$	P
	5.	$\neg\, p(x, v) \vee \neg\, p(v, x)$	1, 2, R, $\{u/x,\ w/x\}$
	6.	$\neg\, p(b, a)$	3, 5, R, $\{x/a,\ v/b\}$
	7.	$\square$	4, 6, R, $\{\ \}$.
		QED	

So we have a refutation. Thus we can conclude that the properties of irreflexive and transitive imply antisymmetry.

end example

example **9.16 The Family Tree Problem**

Suppose we let p stand for the isParentOf relation and let g stand for the isGrandParentOf relation. Then we can define g in terms of p by the following wff, which we'll call G:

$$G = \forall x\ \forall y\ \forall z\ (p(x, z) \wedge p(z, y) \rightarrow g(x, y)).$$

In other words, if x is a parent of z and z is a parent of y, then we conclude that x is a grandparent of y. Suppose we have the following facts about parents, where the letters a, b, c, d, and e denote the names of people:

$$p(a, b),\ p(c, b),\ p(b, d),\ p(a, e).$$

Now, suppose someone claims that $g(a, d)$ is implied by the given facts. Let P denote the conjunction of parent facts.

$$P = p(a, b) \wedge p(c, b) \wedge p(b, d) \wedge p(a, e).$$

So the claim is that the wff W is valid, where

$$W = P \wedge G \rightarrow g(a, d).$$

To prove the claim using resolution, we must prove that $\neg W$ is unsatisfiable. We can observe that $\neg W$ has the following form:

$$\neg W = \neg (P \wedge G \rightarrow g(a, d)) \equiv P \wedge G \wedge \neg g(a, d).$$

We need to put $\neg W$ into clausal form. Since P is a conjunction of atoms, it is already in clausal form. So we need only work on G, which will be in clausal form if we replace the conditional. The result is the clause

$$\neg p(x, z) \vee \neg p(z, y) \vee g(x, y).$$

So the clausal form of $\neg W$ consists of the following six clauses.

$p(a, b)$, $p(c, b)$, $p(b, d)$, $p(a, e)$, $\neg p(x, z) \vee \neg p(z, y) \vee g(x, y)$, and $\neg g(a, d)$.

To do a resolution proof, we start with the six clauses as premises. Our goal is to construct resolvants to obtain the empty clause. Each resolution step includes the most general unifier used for that application of resolution.

Proof:	1.	$p(a, b)$	P
	2.	$p(c, b)$	P
	3.	$p(b, d)$	P
	4.	$p(a, e)$	P
	5.	$\neg p(x, z) \vee \neg p(z, y) \vee g(x, y)$	P
	6.	$\neg g(a, d)$	P
	7.	$\neg p(a, z) \vee \neg p(z, d)$	$5, 6, R, \{x/a, y/d\}$
	8.	$\neg p(b, d)$	$1, 7, R, \{z/b\}$
	9.	$\square$	$3, 8, R, \{\ \}$.
		QED	

So we have a refutation. Therefore, we can conclude that $g(a, d)$ is implied from the given facts.

end example

example 9.17 Diagonals of a Trapezoid

We'll give a resolution proof that the alternate interior angles formed by a diagonal of a trapezoid are equal. This problem is from Chang and Lee [1973]. Let $t(x, y, u, v)$ mean that x, y, u, and v are the four corner points of a trapezoid in clockwise order. Let $p(x, y, u, v)$ mean that edges xy and uv are parallel lines. Let $e(x, y, z, u, v, w)$ mean that angle xyz is equal to angle uvw. We'll assume the following two axioms about trapezoids.

Axiom 1: $\forall x\ \forall y\ \forall u\ \forall v\ (t(x, y, u, v) \rightarrow p(x, y, u, v))$.

Axiom 2: $\forall x\ \forall y\ \forall u\ \forall v\ (p(x, y, u, v) \rightarrow e(x, y, v, u, v, y))$.

To prove: $t(a, b, c, d) \rightarrow e(a, b, d, c, d, b)$.

To prepare for a resolution proof, we need to write each axiom in its clausal form. This gives us the following two clauses:

Axiom 1: $\neg\ t(x, y, u, v) \vee p(x, y, u, v)$.

Axiom 2: $\neg\ p(x, y, u, v) \vee e(x, y, v, u, v, y)$.

Next, we need to negate the statement to be proved and put the result in clausal form, which gives us the following two clauses:

$$t\,(a, b, c, d),$$
$$\neg\ e\,(a, b, d, c, d, b).$$

To do a resolution proof, we start with the four clauses as premises. Our goal is to construct resolvants to obtain the empty clause. Each resolution step includes the most general unifier used for that application of resolution.

Proof:			
	1.	$\neg\ t(x, y, u, v) \vee p(x, y, u, v)$	P
	2.	$\neg\ p(x, y, u, v) \vee e(x, y, v, u, v, y)$	P
	3.	$t(a, b, c, d)$	P
	4.	$\neg\ e(a, b, d, c, d, b)$	P
	5.	$\neg\ p(a, b, c, d)$	2, 4, R, $\{x/a,\ y/b,\ v/d,\ u/c\}$
	6.	$\neg\ t(a, b, c, d)$	1, 5, R, $\{x/a,\ y/b,\ u/c,\ v/d\}$
	7.	$\Box$	3, 6, R, $\{\ \}$.
		QED	

end example

9.1.6 Remarks

In the example proofs we didn't follow a specific strategy to help us choose which clauses to resolve. Strategies are important because they may help reduce the searching required to find a proof. Although a general discussion of strategy is beyond our scope, we'll present a strategy in the next section for the special case of logic programming.

The unification algorithm (9.6) is the original version given by Robinson [1965]. Other researchers have found algorithms that can be implemented more efficiently. For example, the paper by Paterson and Wegman [1978] presents a linear algorithm for unification.

There are also other versions of the resolution inference rule. One approach uses two simple rules, called *binary resolution* and *factoring*, which can be used together to do the same job as resolution. Another inference rule, called *paramodulation*, is used when the equality predicate is present to take advantage of substituting equals for equals. An excellent introduction to automatic reasoning is contained in the book by Wos, Overbeek, Lusk, and Boyle [1984].

Another subject that we haven't discussed is automatic reasoning in higher-order logic. In higher-order logic it's undecidable whether a set of atoms can be unified. Still there are many interesting results about higher-order unification and there are automatic reasoning systems for some higher-order logics. For example, in second-order monadic logic (*monadic logic* restricts predicates to at most one argument) there is an algorithm to decide whether two atoms can be unified. For example, if F is a variable that represents a function, then the two atoms $F(a)$ and a can be unified by letting F be the constant function that returns the value a or by letting F be the identity function. The paper by Snyder and Gallier [1989] contains many results on higher-order unification.

Automatic theorem-proving techniques are an important and interesting part of computer science, with applications to almost every area of endeavor. Probably the most successful applications of automatic theorem proving will be interactive in nature, with the proof system acting as an assistant to the person using it. Typical tasks involve such things as finding ways to represent problems and information to be processed by an automatic theorem prover, finding algorithms that make proper choices in performing resolution, and finding algorithms to efficiently perform unification. We'll look at the programming side of theorem proving in the next section.

Exercises

Constructing Clausal Forms

1. Use Skolem's algorithm, if necessary, to transform each of the following wffs into a clausal form.

   a. $(A \wedge B) \vee C \vee D$.
   b. $(A \wedge B) \vee (C \wedge D) \vee (E \rightarrow F)$.
   c. $\exists y\, \forall x\, (p(x, y) \rightarrow q(x))$.
   d. $\exists y\, \forall x\, p(x, y) \rightarrow q(x)$.
   e. $\forall x\, \forall y\, (p(x, y) \vee \exists z\, q(x, y, z))$.
   f. $\forall x\, \exists y\, \exists z\, [(\neg\, p(x, y) \wedge q(x, z)) \vee r(x, y, z)]$.

Resolution with Propositions

2. What is the resolvant of the propositional clause $p \vee \neg\, p$ with itself? What is the resolvant of $p \vee \neg\, p \vee q$ with itself?

3. Find a resolution proof to show that each of the following sets of propositional clauses is unsatisfiable.
   a. $\{A \vee B, \neg A, \neg B \vee C, \neg C\}$.
   b. $\{p \vee q, \neg p \vee r, \neg r \vee \neg p, \neg q\}$.
   c. $\{A \vee B, A \vee \neg C, \neg A \vee C, \neg A \vee \neg B, C \vee \neg B, \neg C \vee B\}$.

Substitutions and Unification

4. Compute the composition $\theta\sigma$ of each of the following pairs of substitutions.
   a. $\theta = \{x/y\}, \sigma = \{y/x\}$.
   b. $\theta = \{x/y\}, \sigma = \{y/x, x/a\}$.
   c. $\theta = \{x/y, y/a\}, \sigma = \{y/x\}$.
   d. $\theta = \{x/f(z), y/a\}, \sigma = \{z/b\}$.
   e. $\theta = \{x/y, y/f(z)\}, \sigma = \{y/f(a), z/b\}$.

5. Use Robinson's unification algorithm to find a most general unifier for each of the following sets of atoms.
   a. $\{p(x, f(y, a), y), p(f(a, b), v, z)\}$.
   b. $\{q(x, f(x)), q(f(x), x)\}$.
   c. $\{p(f(x, g(y)), y), p(f(g(a), z), b)\}$.
   d. $\{p(x, f(x), y), p(x, y, z), p(w, f(a), b)\}$.

6. Use the Martelli-Montanari unification algorithm to find a most general unifier for each of the following sets of atoms.
   a. $\{p(x, f(y, a), y), p(f(a, b), v, z)\}$.
   b. $\{q(x, f(x)), q(f(x), x)\}$.
   c. $\{p(f(x, g(y)), y), p(f(g(a), z), b)\}$.

Resolution in First-Order Logic

7. What is the resolvant of the clause $p(x) \vee \neg p(f(a))$ with itself? What is the resolvant of $p(x) \vee \neg p(f(a)) \vee q(x)$ with itself?

8. Use resolution to show that each of the following sets of clauses is unsatisfiable.
   a. $\{p(x), q(y, a) \vee \neg p(a), \neg q(a, a)\}$.
   b. $\{p(u, v), q(w, z), \neg p(y, f(x, y)) \vee \neg p(f(x, y), f(x, y)) \vee \neg q(x, f(x, y))\}$.
   c. $\{p(a) \vee p(x), \neg p(a) \vee \neg p(y)\}$.
   d. $\{p(x) \vee p(f(a)), \neg p(y) \vee \neg p(f(z))\}$.
   e. $\{q(x) \vee q(a), \neg p(y) \vee \neg p(g(a)) \vee \neg q(a), p(z) \vee p(g(w)) \vee \neg q(w)\}$.

Proving Theorems

9. Prove that each of the following propositional statements is a tautology by using resolution to prove that its negation is a contradiction.
   a. $(A \vee B) \wedge \neg A \to B$.
   b. $(p \to q) \wedge (q \to r) \to (p \to r)$.
   c. $(p \vee q) \wedge (q \to r) \wedge (r \to s) \to (p \vee s)$.
   d. $[(A \wedge B \to C) \wedge (A \to B)] \to (A \to C)$.

10. Prove that each of the following statements is valid by using resolution to prove that its negation is unsatisfiable.
    a. $\forall x\ p(x) \to \exists x\ p(x)$.
    b. $\forall x\ (p(x) \to q(x)) \wedge \exists x\ p(x) \to \exists x\ q(x)$.
    c. $\exists y\ \forall x\ p(x, y) \to \forall x\ \exists y\ p(x, y)$.
    d. $\exists x\ \forall y\ p(x, y) \wedge \forall x\ (p(x, x) \to \exists y\ q(y, x)) \to \exists y\ \exists x\ q(x, y)$.
    e. $\forall x\ p(x) \vee \forall x\ q(x) \to \forall x\ (p(x) \vee q(x))$.

Challenges

11. Let E be any expression, A and B two sets of expressions, and θ, σ, α any substitutions. Prove each of the following statements about composing substitutions.
    a. $E(\theta\sigma) = (E\theta)\sigma$.
    b. $E\epsilon = E$.
    c. $\theta\epsilon = \epsilon\theta = \theta$.
    d. $(\theta\sigma)\alpha = \theta(\sigma\alpha)$.
    e. $(A \cup B)\theta = A\theta \cup B\theta$.

12. Translate each of the following arguments into first-order predicate calculus. Then use resolution to prove that the resulting wffs are valid by proving that the negations are unsatisfiable.
    a. All computer science majors are people. Some computer science majors are logical thinkers. Therefore, some people are logical thinkers.
    b. Babies are illogical. Nobody is despised who can manage a crocodile. Illogical persons are despised. Therefore, babies cannot manage crocodiles.

13. Translate each of the following arguments into first-order predicate calculus. Then use resolution to prove that the resulting wffs are valid by proving the negations are unsatisfiable.
    a. Every dog either likes people or hates cats. Rover is a dog. Rover loves cats. Therefore, some dog likes people.
    b. Every committee member is rich and famous. Some committee members are old. Therefore, some committee members are old and famous.

c. No human beings are quadrupeds. All men are human beings. Therefore, no man is a quadruped.
d. Every rational number is a real number. There is a rational number. Therefore, there is a real number.
e. Some freshmen like all sophomores. No freshman likes any junior. Therefore, no sophomore is a junior.

14. In the proof of Skolem's algorithm it was shown that $s(W) \to W$ is valid, where $s(W)$ is the clausal form of W. Show that the converse is false by considering the following wff:

$$W = \forall x \; \exists y \; (p(x, y) \vee \neg \; p(y, y)).$$

a. Show that W is valid.
b. Find the clausal form of W and show that it is invalid.

15. It was noted after Skolem's algorithm that Skolem's rule could not be used to remove existential quantifiers until after all implications were eliminated and all negations were moved inward. To confirm this, do each of the following exercises, where W is the following wff and C is any wff that does not contain x or y:

$$W = (\exists x \; p(x) \to C) \wedge \exists y \; (p(y) \wedge \neg \; C).$$

a. Show that W is unsatisfiable.
b. Remove the two existential quantifiers from W with Skolem's rule (9.1). Show that the resulting wff is satisfiable.
c. Eliminate $\to$ from W and then apply (9.1) to the wff obtained. Show that the resulting wff is satisfiable.
d. Apply Skolem's algorithm correctly to W and show that the resulting clausal form is unsatisfiable.

9.2 Logic Programming

In this section we'll see how logic programming is related to logic. To start things off we'll give a gentle introduction to the Prolog language with a discussion about family trees. Then we'll see how resolution comes into the picture as a computation device for logic programs. Finally, we'll study a few basic techniques of logic programming.

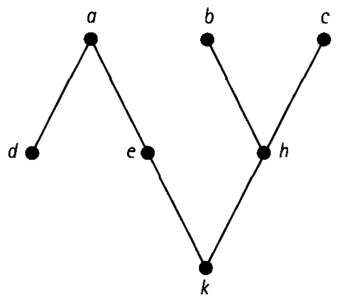


Figure 9.1 A family.

9.2.1 Family Trees

We'll start with a set of parent-child relations. If we let $p(x, y)$ mean "x is a parent of y," then we should start with some parent-child facts. For example, suppose the graph in Figure 9.1 represents a set of parent-child relations, where each node represents the root of a partial family tree with children directly below it and parents directly above it.

For example, d and e are the children of a. We can represent this tree with 6 parent-child facts as follows, where in Prolog each fact ends with a period:

```
p(a,d).
p(a,e).
p(b,h).
p(c,h).
p(e,k).
p(h,k).
```

Finding the Parents of a Person

Suppose that we want to find the parents of k. In Prolog we can find them by typing the following goal or query in response to the interactive prompt |?–, where the uppercase letter X stands for a variable.

```
|?- p(X, k).
```

The computation will search the program facts trying to match $p(X, k)$ with some fact in the program. In this case, the goal matches $p(e, k)$ and the system responds with

```
X = e ?
```

At this point, we either stop the computation or ask it to continue. If we stop it, then most systems simply answer yes, indicating that an answer was found.

If we continue, then the system will search for another match for the goal atom $p(X, k)$. In this case, the goal matches $p(h, k)$ and the system responds with

$$X = h\ ?$$

If we ask it to continue, it will search for another match. But there are none, so the system answers no. So we conclude that e and h are the parents of k.

Finding the Grandparents of a Person

Now suppose we want to find some grandparent relations. To do this we can let $g(x, y)$ mean "x is a grandparent of y." From our knowledge of family relations we know that x is a grandparent of y if there is some z such that x is a parent of z and z is a parent of y. So we can define the isGrandParentOf relation g in terms of the isParentOf relation:

$$g(x, y) \text{ if } p(x, z) \text{ and } p(z, y).$$

In Prolog, we represent this relationship as the following expression, where the variables start with uppercase letters:

$$g(X, Y) :\!-\ p(X, Z),\ p(Z, Y).$$

With the addition of this clause, the Prolog program now looks like the following collection of statements.

```
p(a,d).
p(a,e).
p(b,h).
p(c,h).
p(e,k).
p(h,k).
g(X,Y) :-p(X,Z), p(Z,Y).
```

Suppose that we want to find the grandparents of k. We can find them by typing the following goal or query.

$$|?\text{-}\ g(U, k).$$

The system will search the program statements trying to match $g(U, k)$ with a fact or the left part of a clause. In this case, it matches $g(X, Y)$ with the unifer $\{U/X, Y/k\}$. Now this unifier is applied to the antecedents $p(X, Z)$ and $p(Z, Y)$ to obtain the two new goals

$$p(X, Z),\ p(Z, k).$$

These atoms have to be unified with some facts before the answer yes can be returned. So the system searches for two facts to match the two atoms. One such match to be found is the pair

$$p(a, e),\ p(e, k).$$

with the unifier $\{X/a,\ Z/e\}$. Then the composition of the two unifiers is applied to U:

$$U\ \{U/X,\ Y/k\}\ \{X/a,\ Z/e\} = \mathrm{X}\ \{X/a,\ Z/e\} = a.$$

So the system responds with

$$U = a\ ?$$

If we continue, the system will find a match for the pair $p(b,h), p(h,k)$ with the unifier $\{X/b, Z/h\}$. So the computation responds with

$$U = b\ ?$$

If we continue, the system will find a match for the pair $p(c,h), p(h,k)$ with the unifier $\{X/c, Z/h\}$. So the computation responds with

$$U = c\ ?$$

If we continue the computation, the system will return the answer no because there are no other grandparents listed for k.

Of course we're only touching the surface of the kind of information that we can extract from the parent-child relations. For example, we might want to know answers to questions regarding ancestors, descendants, cousins, and so on. We'll see later that it is an easy matter to define rules to allow us to answer many such questions.

Now that we have a little knowledge of Prolog, in the next few paragraphs we'll define what a logic program is, and we'll show how logic program computations are performed.

9.2.2 Definition of a Logic Program

A *logic program* is a set of clauses, where each clause has exactly one positive literal (i.e., an atom) and zero or more negative literals. Such clauses have one of the following two forms, where A, B_1, ..., B_n are atoms:

A	(one positive and no negative literals)
$A \vee \neg B_1 \vee \cdots \vee \neg B_n$	(one positive and some negative literals).

Notice how we can use equivalences to write a clause as a conditional:

$$A \vee \neg B_1 \vee \cdots \vee \neg B_n \equiv A \vee \neg (B_1 \wedge \cdots \wedge B_n) \equiv B_1 \wedge \cdots \wedge B_n \to A.$$

Such a clause is denoted by writing it backwards as the following expression, where the conjunctions are replaced by commas.

$$A \leftarrow B_1,\ \ldots,\ B_n.$$

We can read this clause as "A is true if B_1, ..., B_n are all true." The atom A on the left side of the arrow is called the *head* of the clause and the atoms on the right form the *body* of the clause. A clause consisting of a single atom A can be read as "A is true." Such a clause is called a *fact* or a *unit* clause. It is a clause without a body.

Goals or Queries

Since a logic program is a set of clauses, we can ask whether anything can be inferred from the program by letting the clauses be premises. In fact, to execute a logic program we must give it a *goal* or *query*, which is a clause of the following form, where $B_1, \ldots, B_n$ are atoms.

$$\leftarrow B_1, \ldots, B_n.$$

It is a clause without a head. We can read the goal as "Are $B_1, \ldots, B_n$ inferred by the program?"

We should note that the clauses in logic programs are often called *Horn* clauses.

example 9.18 The Result of an Exectution

Let P be the logic program consisting of the following three clauses:

$$\begin{array}{l} q(a). \\ r(a). \\ p(x) \leftarrow q(x), r(x). \end{array}$$

Suppose we execute P with the following goal or query:

$$\leftarrow p(a).$$

We can read the query as "Is $p(a)$ true?" or "Is $p(a)$ inferred from P?" The answer is yes. We can argue informally. The two program facts tell us that $q(a)$ and $r(a)$ are both true. The program clause tells us that $p(x)$ is true if $q(x)$ and $r(x)$ are both true. So we infer that $p(a)$ is true by modus ponens. In what follows we'll see how the answer follows from resolution.

end example

9.2.3 Resolution and Logic Programming

Let's see why things are set up to use resolution. First of all, a logic program is a set of clauses, so it is already in the proper form for using resolution. To execute a logic program we need a goal. So suppose P is a logic program and we execute P with the following goal:

$$\leftarrow B_1, \ldots, B_n.$$

For this discussion we'll denote the existential closure of $B_1 \wedge \cdots \wedge B_n$ by

$$\exists(B_1 \wedge \cdots \wedge B_n).$$

With this notation, we read the goal $\leftarrow B_1, \ldots, B_n$ as

"Is $\exists(B_1 \wedge \cdots \wedge B_n)$ inferred by the program P?"

In other words, the goal asks if there a proof of $\exists(B_1 \wedge \cdots \wedge B_n)$ using the clauses of P as premises. Equivalently, the goal asks if there is a proof that the set $P \cup \{\neg \exists(B_1 \wedge \cdots \wedge B_n)\}$ is unsatisfiable.

Now we're getting somewhere because resolution is used to prove unsatisfiablity. Now let's notice that $\neg \exists(B_1 \wedge \cdots \wedge B_n)$ can be written in the following form:

$$\neg \exists(B_1 \wedge \cdots \wedge B_n) \equiv \forall \neg (B_1 \wedge \cdots \wedge B_n) \equiv \forall(\neg B_1 \vee \cdots \vee \neg B_n).$$

Now, as luck would have it, the wff $\forall(\neg B_1 \vee \cdots \vee \neg B_n)$ is none other than a clause, where $\forall$ denotes universal closure. So the goal $\leftarrow B_1, \ldots, B_n$ becomes the following statement about a set of clauses:

"Is there a proof that the set $P \cup \{\forall(\neg B_1 \vee \cdots \vee \neg B_n)\}$ is unsatisfiable?"

As with the other clauses, we'll delete $\forall$ from the notation for the clause. So the goal $\leftarrow B_1, \ldots, B_n$ becomes the following statement about a set of clauses.

"Is there a proof that the set $P \cup \{(\neg B_1 \vee \cdots \vee \neg B_n)\}$ is unsatisfiable?"

Now the goal $\leftarrow B_1, \ldots, B_n$ is just notation for the clause $(\neg B_1 \vee \cdots \vee \neg B_n)$. So the statement can be phrased strictly in terms of logic program notation as

"Is there a proof that the set $P \cup \{\leftarrow B_1, \ldots, B_n\}$ is unsatisfiable?"

Answers to Goals

When we give a goal to a logic program, we usually want more than just the answer yes or no to the goal question. If the answer is yes, we may want to know the values of any variables that appear in the goal. The nice thing about resolution is that the unifiers constructed during the proof provide values for the variables. So we really can read the goal question as "Is there a substitution θ for the variables of $B_1, \ldots, B_n$ such that $(B_1 \wedge \cdots \wedge B_n)\theta$ is inferred by the program P?"

Let's look at an example to see how the notation for logic program clauses makes it easy to find answers to goal questions.

example 9.19 Answering a Goal Question

Let P be the following logic program.

$$q(a).$$
$$p(f(x)) \leftarrow q(x).$$

Suppose also that we have the following goal question:

$$\leftarrow p(y).$$

This means that we want an answer to the question, "Is there a substitution θ such that $p(y)\theta$ is inferred from P?" Let's give the answer first and then see how we got it. The answer is yes. Letting $\theta = \{y/f(a)\}$, we can evaluate $p(y)\theta$ as follows:

$$p(y)\theta = p(y)\{y/f(a)\} = p(f(a)).$$

We claim that $p(f(a))$ is inferred from P. Let's give a resolution proof showing that $P \cup \{\neg\, p(y)\}$ is unsatisfiable. First we'll write the two program clauses and the goal clause in the clausal form needed for resolution. Then we'll write them as premises and start the process of finding a refutation using resolution.

Proof:			
	1.	$q(a)$	P (program clause $q(a)$)
	2.	$p(f(x)) \vee \neg\, q(x)$	P (program clause $p(f(x)) \leftarrow q(x)$)
	3.	$\neg\, p(y)$	P (goal clause $\leftarrow p(y)$)
	4.	$\neg\, q(x)$	2, 3, R, $\{y/f(x)\}$
	5.	$\square$	1, 4, R, $\{x/a\}$.
		QED	

Therefore, by the resolution theorem, $P \cup \{\neg\, p(y)\}$ is unsatisfiable. So the answer to the goal question is yes. But what value of y does the job? We can find it by applying the composition of the mgu's to y as follows:

$$y\ \{y/f(x)\}\ \{x/a\} = f(x)\ \{x/a\} = f(a).$$

Therefore, $p(f(a))$ is a logical consequence of program P.

end example

SLD-Resolution

There are three advantages to the notation used for logic programs.

1. The notation is easy to write down because we don't have to use the symbols $\neg$, $\wedge$, and $\vee$.

2. The notation allows us to interpret a program in two different ways. For example, suppose we have the clause $A \leftarrow B_1, \ldots, B_n$. This clause has the usual logical interpretation "A is true if $B_1, \ldots, B_n$ are all true." The clause also has the procedural interpretation "A is a procedure that is executed by executing the procedures $B_1, \ldots, B_n$ in the order they are written." Most logic programming systems allow this procedural interpretation.

3. The notation makes it easy to apply the resolution rule. We'll discuss this next.

Whenever we apply the resolution rule, we have to do a lot of choosing. We have to choose two clauses to resolve, and we have to choose literals to "cancel" from each clause. Since there are many choices, it's easy to understand why we can come up with many different proof sequences. When resolution is used with logic program clauses, we can specialize the rule.

The specialized rule always picks one clause to be the most recent line of the proof, which is always a goal clause. Start the proof by picking the initial goal. Select the leftmost atom in the goal clause as the literal to "cancel." For the second clause, pick a program clause whose head unifies with the atom selected from the goal clause. The resolvant of these two clauses is created by first replacing the leftmost atom in the goal clause by the body atoms of the program clause and then applying the unifier to the resulting goal. Here is a formal description of the rule, which is called the *SLD-resolution* rule:[1]

SLD-Resolution Rule **(9.11)**

To resolve the goal $\leftarrow B_1, \ldots, B_k$ with the clause $A \leftarrow A_1, \ldots, A_n$, perform the following steps:

1. Unify B_1 and A and obtain most general unifier θ.
2. Replace B_1 by the body atoms $A_1, \ldots, A_n$.
3. Apply θ to the result to obtain the resolvant

$$\leftarrow (A_1, \ldots, A_n, B_2, \ldots, B_k)\theta.$$

Constructing a Logic Program Proof

To construct a logic program proof, we start by listing each program clause as a premise. Then we write the goal clause as a premise. Now we use (9.11) repeatedly to add new resolvants to the proof, each new resolvant being constructed from the goal on the previous line together with some program clause. We can summarize the application of (9.11) with the following four-step procedure:

1. Pick the goal clause on the last line of the partial proof, and select its leftmost atom, say B_1.
2. Find a program clause whose head unifies with B_1, say by θ. Be sure the two clauses have distinct sets of variables (rename if necessary).
3. Replace B_1 in the goal clause with the body of the program clause.
4. Apply θ to the goal constructed on line 3 to get the resolvant, which is placed on a new line of the proof.

[1]SLD-resolution means selective linear resolution of definite clauses. In our case we always "select" the leftmost atom of the goal clause.

The Family Tree Revisited

We'll introduce the use of the SLD-resolution rule by revisiting family tree relations. Suppose we are given the following logic program, where p means isParentOf and g means isGrandparentOf.

$$\begin{aligned}&p(a,b).\\&p(d,b).\\&p(b,c).\\&g(x,y) \leftarrow p(x,z), p(z,y).\end{aligned}$$

We'll execute the program by giving it the following goal question:

$$\leftarrow g(w,\ c).$$

Since there is a variable w in this goal, we can read the goal as the question

"Is there a grandparent for c?"

The resolution proof starts by letting the program clauses and the goal clause be premises. For this example we have the following five lines:

Proof:	1.	$p(a,\ b)$	P
	2.	$p(d,\ b)$	P
	3.	$p(b,\ c)$	P
	4.	$g(x,\ y) \leftarrow p(x,\ z),\ p(z,\ y)$	P
	5.	$\leftarrow g(w,\ c)$	P Initial goal

The proof starts by resolving the initial goal on line 5 with some program clause. The atom $g(w,\ c)$ from the initial goal unifies with $g(x,\ y)$, the head of the program clause on line 4, by the mgu

$$\theta_1 = \{w/x,\ y/c\}.$$

Therefore, we can use (9.11) to resolve the two clauses on lines 4 and 5. So we replace the goal atom $g(w,\ c)$ on line 5 with the body of the clause on line 4 and then apply the mgu θ_1 to the result to obtain the following resolvant.

$$p(x,\ z),\ p(z,\ c).$$

Let's compare what we've just done for logic program clauses using (9.11) to the case for first-order clauses using (9.8). The following two lines are copies of lines 4 and 5 in which we've included the clausal notation for each logic program clause:

Logic Program Notation	*Clausal Notation*
4. $g(x,\ y) \leftarrow p(x,\ z),\ p(z,\ y)$	$g(x,\ y) \vee \neg\, p(x,\ z) \vee \neg\, p(z,\ y)$
5. $\leftarrow g(w,\ c)$	$\neg\, g(w,\ c)$

We apply (9.11) to the logic program notation clauses, and we apply (9.8) to the clauses in clausal notation. This gives the following resolvant.

Logic Program Notation	*Clausal Notation*
$\leftarrow p(x, z), p(z, c)$	$\neg\, p(x, z) \vee \neg\, p(z, c)$

So we get the same answer with either method.

Now let's continue the proof. We'll write down the new resolvant on line 6 of our proof, in which we've added the mgu to the reason column:

$$6.\ \leftarrow p(x, z), p(z, c) \qquad 4, 5, R, \theta_1 = \{w/x, y/c\}$$

To continue the proof according to (9.11), we must choose this new goal on line 6 for one of the clauses, and we must choose its leftmost atom $p(x, z)$ for "cancellation." For the second clause we'll choose the clause on line 1 because its head $p(a, b)$ unifies with our chosen atom by the mgu

$$\theta_2 = \{x/a, z/b\}.$$

To apply (9.11), we must replace $p(x, z)$ on line 6 by the body of the clause on line 1 and then apply θ_2 to the result. Since the clause on line 1 does not have a body, we simply delete $p(x, z)$ from line 6 and apply θ_2 to the result, obtaining the resolvant $\leftarrow p(b, c)$. So we have a new goal:

$$\leftarrow p(b, c).$$

Let's compute this result in terms of both (9.11) and (9.8). The clauses on lines 1 and 6 take the following forms, in which we've added the regular clausal notation for each clause.

Logic Program Notation	*Clausal Notation*
1. $p(a, b)$	$p(a, b)$
6. $\leftarrow p(x, z), p(z, c)$	$\neg\, p(x, z) \vee \neg\, p(z, c)$

After applying (9.11) and (9.8) to the respective notations on lines 1 and 6, we obtain the following resolvant:

Logic Program Notation	*Clausal Notation*
$\leftarrow p(b, c)$	$\neg\, p(b, c)$

So we can continue the proof by writing down the new resolvant on line 7.

$$7.\ \leftarrow p(b, c) \qquad 1, 6, R, \theta_2 = \{x/a, z/b\}$$

To continue the proof using (9.11), we must choose the goal on line 7 together with its only atom $p(b, c)$. It unifies with the head $p(b, c)$ of the clause on line 3 by the empty unifier

$$\theta_3 = \{\ \}.$$

Since there is only one atom in the goal clause of line 7 and there is no body in the clause on line 3, it follows that the resolvant of the clauses on these two lines is just the empty clause. So our proof is completed by writing this information on line 8.

8.	$\square$	3, 7, R, $\theta_3 = \{\ \}$.
	QED	

To finish things off, we'll collect the eight steps of the proof and rewrite them as a single unit:

Proof:	1.	$p(a, b)$	P
	2.	$p(d, b)$	P
	3.	$p(b, c)$	P
	4.	$g(x, y) \leftarrow p(x, z), p(z, y)$	P
	5.	$\leftarrow g(w, c)$	P Initial goal
	6.	$\leftarrow p(x, z), p(z, c)$	4, 5, R, $\theta_1 = \{w/x, y/c\}$
	7.	$\leftarrow p(b, c)$	1, 6, R, $\theta_2 = \{x/a, z/b\}$
	8.	$\square$	3, 7, R, $\theta_3 = \{\ \}$.
		QED	

Since $\square$ was obtained, we have a refutation. So the answer is yes to the goal

$$\leftarrow g(w, c).$$

In other words, "Yes, there is a grandparent w of c." But we want to know more. We want to know the value of w that gives a yes answer. We can recover the value by composing the three unifiers θ_1, θ_2, and θ_3 and then applying the result to w:

$$w\theta_1\theta_2\theta_3 = a.$$

So the goal question "Is there a grandparent w of c?" is answered:

$$\text{Yes,}$$
$$w = a.$$

What about other possibilities? By looking at the facts, we notice for this example that d is also a grandparent of c. Can this answer be computed? Sure. Keep the first six lines of the proof as they are. Then resolve the goal on line 6 with the clause on line 2. The goal atom $p(x, z)$ on line 6 unifies with the head $p(d, b)$ from line 1 by mgu

$$\theta_2 = \{x/d, z/b\}.$$

This θ_2 is different from the previous θ_2. So we get a new line 7 and the same line 8 to obtain the following alternative refutation.

7.	$\leftarrow p(b, c)$	2, 6, R, $\theta_2 = \{x/d, z/b\}$
8.	$\square$	3, 7, R, $\theta_3 = \{\ \}$.
	QED	

With this proof we obtain the following new value for w:

$$w\theta_1\theta_2\theta_3 = d.$$

So the goal question "Is there a grandparent w of c?" can also be answered:

$$\text{Yes,}$$
$$w = d.$$

We can observe from the facts that a and d are the only grandparents of c, and we've come up with refutations to calculate them. So it's time to see whether a computation can actually come up with both answers.

Computation Trees

Now that we have an example under our belts, let's look again at the general picture. The preceding proof had two possible yes answers. We would like to find a way to represent all possible answers (i.e., proof sequences) for a goal. For our purposes a tree will do the job.

A *computation tree* for a goal is an ordered tree whose root is the goal. The children of any parent node are all the possible goals (i.e., resolvants) that can be obtained by resolving the parent goal with a program clause. We agree to order the children of each node from left to right in terms of the top-to-bottom ordering of the program clauses that are used with the parent to create the children. Each parent-child branch is labeled with the mgu obtained to create the child. A leaf may be the empty clause or a goal. If the empty clause occurs as a leaf, we write "yes" together with the values of any variables that occur in the original goal at the root of the tree. If a goal occurs as a leaf, this means that it can't be resolved with any program clause, so we write "failure." The computation tree will always show all possible answers for the given goal at its root.

For example, the computation tree for the goal $g(w, c)$ with respect to our example program can be pictured as shown in Figure 9.2. Notice that the tree contains all possible answers to the goal question.

A logic programming system needs a strategy to search the computation tree for a leaf with a yes answer. The strategy used by most Prolog systems is the *depth-first* search strategy, which starts by traversing the tree down to the leftmost leaf. If the leaf is the empty clause, then the yes answer is reported. If the leaf is a failure leaf, then the search returns to the parent of the leaf. At this point a depth-first search is started at the next child to the right. If there is no next child, then the search returns to the parent of the parent, and a depth-first search starts with its next child to the right, and so on. If this process eventually

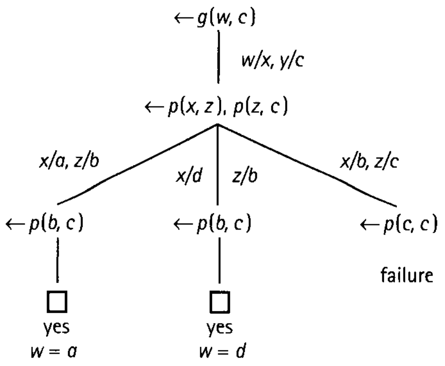


Figure 9.2 Computation tree.

returns to the root of the tree and there are no more paths to search, then failure is reported.

It might be desirable for a logic programming system to attempt to find all possible answers to a goal question. One strategy for attempting to find all possible answers is called *backtracking.* For example, with depth-first search we perform backtracking by continuing the depth-first search process from the point at which the last yes answer was found. In other words, when a yes answer is found, the system reports the answer and then continues just as though a failure leaf was encountered.

In the next few examples we'll construct some computation trees and discuss the problems that can arise in trying to find all possible answers to a goal question.

example 9.20 Many Possible Answers

Let's consider the following two-clause program:

$$p(a).$$
$$p(\text{succ}(x)) \leftarrow p(x).$$

Suppose we give the following goal to the program:

$$\leftarrow p(x).$$

This goal will resolve with either one of the program clauses. So the root of the computation tree has two children. One child, the empty clause, results from the resolution of goal $\leftarrow p(x)$ with the fact $p(a)$. The other child results from the resolution of goal $\leftarrow p(x)$ with the clause $p(\text{succ}(x)) \leftarrow p(x)$.

But before this happens, we need to change variables. We'll replace x by x_1 in the program clause to obtain $p(\text{succ}(x_1)) \leftarrow p(x_1)$. Now resolving the goal $\leftarrow p(x)$ with this clause produces the goal $\leftarrow p(x_1)$, which becomes the second child of the root.

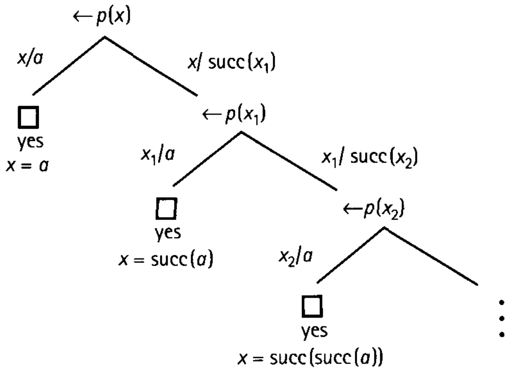


Figure 9.3 Infinitely many answers.

The process starts all over again with the goal $\leftarrow p(x_1)$. To keep track of variable names, we'll replace x by x_2 in the second program clause. Then resolve the goal $\leftarrow p(x_1)$ with $p(\text{succ}(x_2)) \leftarrow p(x_2)$ to obtain the goal $\leftarrow p(x_2)$. This process continues forever.

The computation tree for this example is shown in Figure 9.3. It is an infinite tree that continues the indicated pattern forever. If we use the depth-first search rule, the first answer is "yes, $x = a$." If we force backtracking, the next answer we'll get is "yes, $x = \text{succ}(a)$." If we force backtracking again, we'll get the answer "yes, $x = \text{succ}(\text{succ}(a))$." Continuing in this way, we can generate the following infinite sequence of possible values for x:

$$a, \text{succ}(a), \text{succ}(\text{succ}(a)), \ldots, \text{succ}^k(a), \ldots.$$

end example

example **9.21 Two Possible Answers**

Consider the following three-clause program, in which the third clause has more than one atom in its body.

$$q(a).$$
$$p(a).$$
$$p(f(x)) \leftarrow p(x), q(x).$$

Figure 9.4 shows a few levels of the computation tree for the goal $\leftarrow p(x)$. Notice that as we travel down the rightmost path from the root, the number of goal atoms at each node is increased by one for each new level.

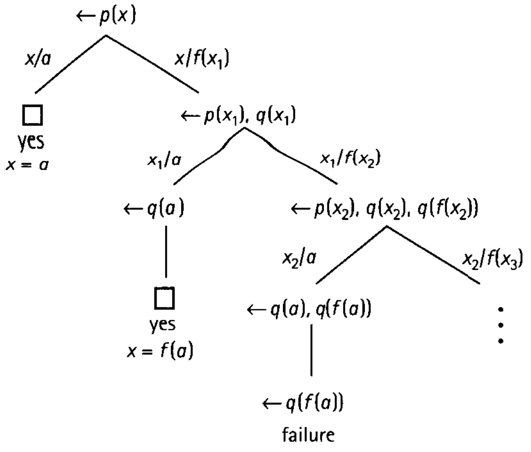


Figure 9.4 Only two answers.

Using the depth-first search rule, we obtain the answer "yes, $x = a$." Backtracking works one time to give the answer "yes, $x = f(a)$." If we force backtracking again, then the computation takes an infinite walk down the tree, failing at each leaf.

end example

example **9.22 How to Miss Many Answers**

Suppose we're given the following three-clause program.

$$\begin{aligned} &p(f(x)) \leftarrow p(x). \\ &p(a). \\ &p(b). \end{aligned}$$

We'll start with the goal

$$\leftarrow p(x).$$

The computation tree will be a ternary tree because there are three "p" clauses that match each goal. Figure 9.5 shows the first few levels of the tree. The tree is infinite, and there are infinitely many yes answers to the goal question. The infinite sequence of possible values for x is

$$a,\ b,\ f(a),\ f(b),\ f(f(a)),\ f(f(b)), \ldots,\ f^k(a),\ f^k(b), \ldots.$$

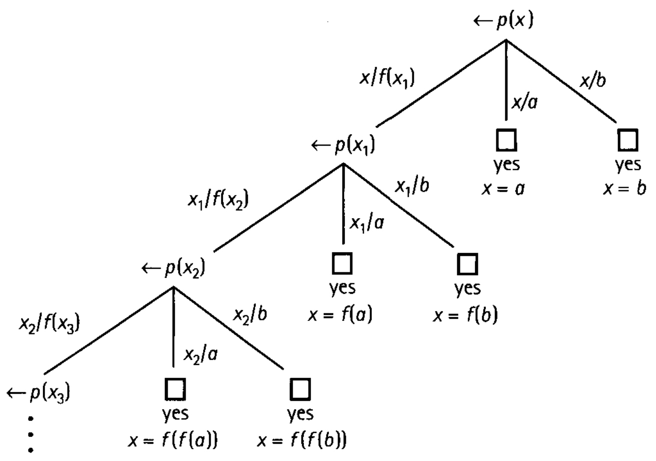


Figure 9.5 Many answers to miss.

Notice that if we used the depth-first search strategy, then the computation would take an infinite walk down the left branch of the tree. So although there are infinitely many answers, the depth-first search strategy won't find even one of them.

end example

In the preceding example, depth-first search did not find any answers to the goal $\leftarrow p(x)$. Suppose we reordered the three program clauses as follows:

$$p(a).$$
$$p(b).$$
$$p(f(x)) \leftarrow p(x).$$

The computation tree corresponding to these three clauses can be searched in a depth-first fashion with backtracking to generate all the answers to the goal $\leftarrow p(x)$. Suppose we write the three clauses in the following order:

$$p(a).$$
$$p(f(x)) \leftarrow p(x).$$
$$p(b).$$

The computation tree for these three clauses, when searched with depth-first and backtracking, will yield some, but not all, of the possible answers.

So when a logic programming language uses depth-first search, two problems can occur when the computation tree for a goal is infinite:

1. The yes answers found may depend on the order of the clauses.

2. Backtracking might not find all possible yes answers to a goal.

Many logic programming systems use the depth-first search strategy because it's efficient to implement and because it reflects the procedural interpretation of a clause. For example, the clause $A \leftarrow B$, C represents a procedure named A that is executed by first calling procedure B and then calling procedure C.

Another search strategy is called *breadth-first search*. It looks for a yes answer by examining all the children of the root. Then it looks at all nodes at the next level of the tree, and so on. This strategy will find all possible answers to a goal question.

Some implementation strategies for searching the computation tree use breadth-first search with a twist. All children of a node are searched in parallel. A search at a particular node is started only when the goal atom has not already occurred at a higher level in the tree. If the goal atom matches a goal at a higher level in the tree, then the process waits for the answer to the other goal. When it receives the answer, then it continues with its search. This technique requires a table containing previous goal atoms and answers. It has proved useful in detecting certain kinds of loops that give rise to infinite computation trees. In some cases the search process won't take an infinite walk. An introduction to these ideas is given in Warren [1992].

9.2.4 Logic Programming Techniques

Let's spend some time discussing a few elementary techniques to construct logic programs. First we'll see how to construct logic programs that process relations. Then we'll discuss logic programs that process functions. The clauses in our examples are ordered to take advantage of the depth-first search strategy. This strategy is used by most Prolog systems.

A Technique for Relations

Logic programming allows us to easily process many relations because relations are just predicates. For example, suppose that we want to write the isAncestorOf relation in terms of the isParentOf relation, where an ancestor is either a parent, or a grandparent, or a great-grandparent, and so on. The next example discusses a technique for solving this type of problem.

example 9.23 Acyclic Transitive Closure

The isAncestorOf relation is the transitive closure of the isParentOf relation. In general terms, suppose we're given a binary relation r whose graph is acyclic (i.e., there are no cycles) and we need to compute the transitive closure of r. If we let tc denote the transitive closure of r, the following two-clause program does the job:

$$\mathrm{tc}(x, y) \leftarrow r(x, y)$$
$$\mathrm{tc}(x, y) \leftarrow r(x, z),\ \mathrm{tc}(z, y).$$

For example, suppose r is the isParentOf relation. Then tc is the isAncestorOf relation. The first clause can be read as "x is an ancestor of y if x is a parent of y," and the second clause can be read as "x is an ancestor of y if x is a parent of z and z is an ancestor of y."

end example

A Technique for Computing Functions

Now let's see whether we can find a technique to construct logic programs to compute functions. Actually, it's pretty easy. The major thing to remember in translating a function definition to a logic definition is that a functional equation like

$$f(x) = y$$

can be represented by a predicate expression such as

$$\text{pf}(x,\ y).$$

The predicate name "pf" can remind us that we have a "predicate for f." The predicate expression pf(x, y) can still be read as "f of x is y."

Now let's discuss a technique to construct a logic program for a recursively defined function. If f is defined recursively, then there is at least one part of the definition that defines $f(x)$ in terms of some $f(y)$. In other words, some part of the definition of f has the following form, where $E(f(y))$ denotes an expression containing $f(y)$:

$$f(x) = E(f(y)).$$

Using our technique to create a predicate for this functional equation, we get the following expression:

$$\text{pf}(x,\ E(f(y))).$$

But we aren't done yet because the recursive definition of f causes $f(y)$ to occur as an argument in the predicate. Since we're trying to compute f by the predicate pf, we need to get rid of $f(y)$. The solution is to replace $f(y)$ by a new variable z. We can represent this replacement by writing down the following version of the expression:

$$\text{pf}(x,\ E(z)) \text{ where } z = f(y).$$

Now we have a functional equation $z = f(y)$, which we can replace by pf(y, z). So we obtain the following expression:

$$\text{pf}(x,\ E(z)) \text{ where pf}(y,\ z).$$

The transformation to a logic program is now simple: Replace the word "where" by the symbol $\leftarrow$ to obtain a logic program clause as follows:

$$\text{pf}(x,\ E(z)) \leftarrow \text{pf}(y,\ z).$$

Thus we have a general technique to transform a functional equation into a logic program. Here are the steps, all in one place:

$f(x) = E(f(y))$	The given functional equation.
$\text{pf}(x, E(f(y)))$	Create a predicate expression.
$\text{pf}(x, E(z))$ where $z = f(y)$	Let $z = f(y)$.
$\text{pf}(x, E(z))$ where $\text{pf}(y, z)$	Create a predicate expression.
$\text{pf}(x, E(z)) \leftarrow \text{pf}(y, z)$	Create a clause.

Of course, there may be more work to do depending on the complexity of the expression $E(z)$. Let's do some examples to help get the look and feel of this process.

example

9.24 The Factorial Function

Suppose we want to write a logic program to compute the factorial function. Letting $f(x) = x!$, we have the following recursive definition of f:

$$f(0) = 1$$
$$f(x) = x * f(x-1).$$

To implement f as a logic program, we'll let "fact" be the predicate to compute f. Then the two equations of the recursive definition become the following two predicate expressions:

$$\text{fact}(0, 1)$$
$$\text{fact}(x, x * f(x-1)).$$

The second statement contains the argument $f(x-1)$, which we'll replace by a new variable y to obtain the following version of the two expressions:

$$\text{fact}(0, 1)$$
$$\text{fact}(x, x * y) \text{ where } y = f(x-1).$$

Now we can change the functional equation $y = f(x-1)$ into a predicate expression to obtain the following version:

$$\text{fact}(0, 1)$$
$$\text{fact}(x, x * y) \text{ where } \text{fact}(x-1, y).$$

Therefore, the desired logic program has the following two clauses:

$$\text{fact}(0, 1)$$
$$\text{fact}(x, x * y) \leftarrow \text{fact}(x-1, y).$$

end example

example

9.25 The Length Function

Suppose we want to write a logic program to compute the length of a list. Let's start with the following recursively defined function L that does the job.

$$\begin{aligned} &L(\langle\ \rangle) = 0 \\ &L(x :: y) = L(y) + 1. \end{aligned}$$

We can start by writing down two predicate expressions to represent these two functional equations. We'll use the predicate name "length" as follows:

$$\begin{aligned} &\text{length}(\langle\ \rangle, 0) \\ &\text{length}(x :: y, L(y) + 1). \end{aligned}$$

The second expression contains an occurrence of the function L, which we're trying to define. So we'll replace $L(y)$ by a new variable z to obtain the following version:

$$\begin{aligned} &\text{length}(\langle\ \rangle, 0) \\ &\text{length}(x :: y, z + 1) \;\text{ where } z = L(y). \end{aligned}$$

Now replace the functional equation $z = L(y)$ by the predicate expression length(y, z) to obtain the following version:

$$\begin{aligned} &\text{length}(\langle\ \rangle, 0) \\ &\text{length}(x :: y, z + 1) \;\text{ where length}(y, z). \end{aligned}$$

Lastly, convert the expressions to the following logic program clauses:

$$\begin{aligned} &\text{length}(\langle\ \rangle, 0). \\ &\text{length}(x :: y, z + 1) \leftarrow \text{length}(y, z). \end{aligned}$$

end example

example

9.26 Deleting an Element

Suppose we want to delete the first occurrence of an element from a list. A recursively defined function to do the job can be written as follows:

$$\begin{aligned} \text{delete}(x, L) = {} & \text{if } L = \langle\ \rangle \text{ then } \langle\ \rangle \\ & \text{else if head}(L) = x \text{ then tail}(L) \\ & \text{else head}(L) :: \text{delete}(x, \text{tail}(L)). \end{aligned}$$

We'll construct a logic program to compute this function. It's much easier to write a logic program for a function described as a set of equations. So we'll write the function as the following three equations:

$$\begin{aligned}&\text{delete}\,(x, \langle\ \rangle) = \langle\ \rangle\\&\text{delete}\,(x, x :: T) = T\\&\text{delete}\,(x, y :: T) = y :: \text{delete}\,(x, T)\,.\end{aligned}$$

First we'll convert each equation to a predicate expression using the predicate named "remove" as follows:

$$\begin{aligned}&\text{remove}\,(x, \langle\ \rangle, \langle\ \rangle)\\&\text{remove}\,(x, x :: T, T)\\&\text{remove}\,(x, y :: T, y :: \text{delete}\,(x, T))\,.\end{aligned}$$

Since the functional value delete(x, T) occurs in the third expression, we'll replace it by a new variable U to obtain the following version:

$$\begin{aligned}&\text{remove}\,(x, \langle\ \rangle, \langle\ \rangle)\\&\text{remove}\,(x, x :: T, T)\\&\text{remove}\,(x, y :: T, y :: U)\ \text{ where } U = \text{delete}\,(x, T)\,.\end{aligned}$$

Now replace the functional equation U = delete(x, T) by the predicate expression remove(x, T, U) as follows:

$$\begin{aligned}&\text{remove}\,(x, \langle\ \rangle, \langle\ \rangle)\\&\text{remove}\,(x, x :: T, T)\\&\text{remove}\,(x, y :: T, y :: U)\ \text{ where remove}\,(x, T, U)\,.\end{aligned}$$

Now transform these three expressions into a three-clause logic program.

$$\begin{aligned}&\text{remove}\,(x, \langle\ \rangle, \langle\ \rangle)\,.\\&\text{remove}\,(x, x :: T, T)\,.\\&\text{remove}\,(x, y :: T, y :: U) \leftarrow \text{remove}\,(x, T, U)\,.\end{aligned}$$

end example

Exercises

Family Trees

1. Suppose you are given an isParentOf relation. Find a definition for each of the following relations.
   a. isChildOf.
   b. isGrandchildOf.
   c. isGreatGrandparentOf.

2. Suppose you are given an isParentOf relation. Try to find a definition for each of the following relations. *Hint:* You might want to consider some kind of test for equality.
   a. isSiblingOf.
   b. isCousinOf.
   c. isSecondCousinOf.
   d. isFirstCousinOnceRemovedOf.

Computation by SLD-Resolution

3. Suppose we're given the following logic program:

$$\begin{aligned}&p(a,b).\\&p(a,c).\\&p(b,d).\\&p(c,e).\\&g(x,y) \leftarrow p(x,z), p(z,y).\end{aligned}$$

   a. Find a resolution proof for the goal $g(a,\ w)$.
   b. Draw a picture of the computation tree for the goal $g(a,\ w)$.

4. Suppose we're given the following logic program:

$$\begin{aligned}&p(a).\\&p(g(x)) \leftarrow p(x).\\&p(b).\end{aligned}$$

   a. Draw at least three levels of the computation tree for the goal $p(x)$.
   b. What are the possible yes answers for the goal $p(x)$?
   c. Describe the values of x that are generated by backtracking with the depth-first search strategy for the goal $p(x)$.

5. The following logic program claims to test an integer to see whether it is a natural number, where pred(x, y) means that the predecessor of x is y:

$$\begin{aligned}&\text{isNat}(0).\\&\text{isNat}(x) \leftarrow \text{isNat}(y),\ \text{pred}(x,y).\end{aligned}$$

   a. What happens when the goal is isNat(2)?
   b. What happens when the goal is isNat(−1)?

Logic Programming

6. Let r denote a binary relation. Write logic programs to compute each of the following relations.
   a. The symmetric closure of r.
   b. The reflexive closure of r.

7. Translate each of the following functional definitions into a logic program. *Hint:* First, translate the if-then-else definitions into equational definitions.
   a. The function f computes the nth Fibonacci number:

      $f(n) =$ if $n = 0$ then 0 else if $n = 1$ then 1 else $f(n-1) + f(n-2)$.
   b. The function "cat" computes the concatenation of two lists:

      $\text{cat}(x, y) =$ if $x = \langle\ \rangle$ then y else $\text{head}(x) :: \text{cat}(\text{tail}(x), y)$.
   c. The function "nodes" computes the number of nodes in a binary tree:

      $\text{nodes}(t) =$ if $t = \langle\ \rangle$ then 0 else $1 + \text{nodes}(\text{left}(t)) + \text{nodes}(\text{right}(t))$.

8. Find a logic program to implement each of the following functions, where the variables represent elements or lists.
   a. equalLists(x, y) tests whether the lists x and y are equal.
   b. member(x, y) tests whether x is an element of the list y.
   c. all(x, y) is the list obtained from y by removing all occurrences of x.
   d. makeSet(x) is the list obtained from x by deleting repeated elements.
   e. subset(x, y) tests whether x, considered as a set, is a subset of y.
   f. equalSets(x, y) tests whether x and y, considered as sets, are equal.
   g. subBag(x, y) tests whether x, considered as a bag, is a subbag of y.
   h. equalBags(x, y) tests whether the bags x and y are equal.

Challenges

9. Suppose we have a schedule of classes with each entry having the form class(i, s, t, p), which means that class i section s meets at time t in place p. Find a logic program to compute the possible schedules available for a given list of classes.

10. Write a logic program to test whether a propositional wff is a tautology. Assume that the wffs use the four operators in the set $\{\neg, \wedge, \vee, \rightarrow\}$. *Hint:* Use the method of Quine together with the fact that if A is a wff containing a propsitional variable p, then A is a tautology iff $A(p/\text{true})$ and $A(p/\text{false})$ are both tautologies. To assist in finding the propositional variables, assume that the predicate atom(x) means that x is a propositional variable.

9.3 Chapter Summary

The major component of automatic reasoning for the first-order predicate calculus is the resolution inference rule. Resolution proofs work by showing that a wff is unsatisfiable. So to prove that a wff is valid, we can use resolution to show that its negation is unsatisfiable. Resolution requires wffs to be represented as sets of clauses, which can be constructed by Skolem's algorithm. Before each step of a resolution proof involving predicates, the unification algorithm must calculate a substitution—a most general unifier—that will unify a set of atoms. The process of applying the resolution rule can be programmed to perform automatic reasoning.

Logic programs consist of clauses that have one positive literal and zero or more negative literals. A logic program goal is a clause consisting of one or more negative literals. Logic program goals are computed by a modification of resolution called SLD-resolution. Each goal of a logic program has an associated computation tree that can be searched in a variety of ways. The depth-first search strategy is used by most logic programming languages. Elementary techniques for logic programming include the implementation of relations and recursively defined functions.

chapter 10

Algebraic Structures and Techniques

Algebraic rules of procedure were proclaimed as if they were divine revelations....

—From *The History of Mathematics* by David M. Burton

The word "algebra" comes from the word "al-jabr" in the title of the textbook *Hisâb al-jabr w'al-muqâbala*, which was written around 820 by the mathematician and astronomer al-Khowârizmî. The title translates roughly to "calculations by restoration and reduction," where restoration—al-jabr—refers to adding or subtracting a number on both sides of an equation, and reduction refers to simplification. We should also note that the word "algorithm" has been traced back to al-Khowârizmî because people used his name—mispronounced—when referring to a method of calculating with Hindu numerals that was contained in another of his books.

Having studied high school algebra, most of us probably agree that algebra has something to do with equations and simplification. In high school algebra we simplified a lot. In fact, we were often given the one word command "simplify" in the exercises. So we tried to somehow manipulate a given expression into one that was simpler than the given one, although this direction was a bit vague, and there always seemed to be a question about what "simplify" meant. We also tried to describe word problems in terms of algebraic equations and then to apply our simplification methods to extract solutions. Everything we did dealt with numbers and expressions for numbers.

In this chapter we'll clarify and broaden the idea of an algebra. The chapter introduces the notions and notations of algebra with special emphasis on the techniques and applications of algebra in computer science.

chapter guide

Section 10.1 introduces the idea of an algebra. We'll see that high school algebra is just one kind of algebra.

Section 10.2 introduces Boolean algebra. We'll discuss some techniques to simplify Boolean expressions, and we'll see how to construct digital circuits.

Section 10.3 introduces the idea of an abstract data type as an algebra. As examples, we'll discuss some properties of the natural numbers, lists, strings, stacks, queues, binary trees, and priority queues.

Section 10.4 introduces relational algebras and functional algebras. We'll see how these ideas are applied to databases and functional programming.

Section 10.5 introduces a collection of algebraic ideas that are useful for computational problems. We'll introduce congruences and see how some results are applied in cryptology. We'll also introduce subalgebras and morphisms.

10.1 What Is an Algebra?

Before we say just what an algebra is, let's see how an algebra is used in the problem-solving process. An important part of problem solving is the process of transforming informal word problems into formal things like equations, expressions, or algorithms. Another important part of problem solving is the process of transforming these formal things into solutions by solving equations, simplifying expressions, or implementing algorithms. For example, in high school algebra we tried to describe certain word problems in terms of algebraic equations, and then we tried to solve the equations. An algebra should provide tools and techniques to help us describe informal problems in formal terms and to help us solve the resulting formal problems.

The Description Problem

How can we describe something to another person in such a way that the person understands exactly what we mean? One way is to use examples. But sometimes examples may not be enough for a proper understanding. It is often useful at some point to try to describe an object by describing some properties that it possesses. So we state the following general problem:

The Description Problem
Describe an object.

Whatever form a description takes, it should be communicated in a clear and concise manner so that examples or instances of the object can be easily checked for correctness. Try to describe one of the following things to a friend:

A car.
The left side of a person.
The number zero.
The concept of area.

Most likely, you'll notice that the description of an object often depends on the knowledge level of the audience.

We need some tools to help us describe properties of the things we are talking about, so we can check not only the correctness of examples, but also the correctness of the descriptions. Algebras provide us with natural notations that can help us give precise descriptions for many things, particularly those structures and ideas that are used in computer science.

High School Algebra

A natural example of an algebra that we all know and love is the algebra of numbers. We learned about it in school, and we probably had different ideas about what it was. First, we learned about arithmetic of the natural numbers $\mathbb{N}$, using the operation of addition. We came eventually to believe things like

$$7 + 12 = 19, \quad 3 + 5 = 5 + 3, \quad \text{and} \quad 4 + (6 + 2) = (4 + 6) + 2.$$

Soon we learned about multiplication, negative numbers, and the integers $\mathbb{Z}$. It seemed that certain numbers like 0 and 1 had special properties such as

$$14 + 0 = 14, \quad 1 \cdot 47 = 47, \quad \text{and} \quad 0 = 9 + (-9).$$

Somewhere along the line, we learned about division, the rational numbers $\mathbb{Q}$, and the fact that we could not divide by zero.

Then came the big leap. We learned to denote numbers by symbols like the letters x and y and by expressions like $x^2 + y$. We spent much time transforming one expression into another, such as $x^2 + 4x + 4 = (x + 2)(x + 2)$. All this had something to do with algebra, perhaps because that was the name of the class.

There are two main ingredients to the algebra that we studied in high school. The first is a set of numbers to work with, such as the real numbers $\mathbb{R}$. The second is a set of operations on the numbers, such as $-$ and $+$. We learned about the general properties of the operations, such as $x + y = y + x$ and $x + 0 = x$. And we learned to use these properties to simplify expressions and solve equations.

Now we are in position to discuss algebra from a more general point of view. We will see that high school algebra is just one of many different kinds of algebras.

10.1.1 Definition of an Algebra

An *algebra* is a structure consisting of one or more sets together with one or more operations on the sets. The sets are often called *carriers* of the algebra. This is a very general definition. If this is the definition of an algebra, how can it help us solve problems? As we will see, the utility of an algebra comes from knowing how to use the operations.

For example, high school algebra is an algebra with the single carrier $\mathbb{R}$, or maybe $\mathbb{Q}$. The operators of the algebra are $+$, $-$, $\cdot$, and $\div$. The constants 0 and 1 are also important to consider because they have special properties. Note that a constant can be thought of as a nullary operation (having arity zero). Many familiar properties hold among the operations, such as the fact that multiplication distributes over addition: $a \cdot (b + c) = a \cdot b + a \cdot c$; and the fact that we can cancel: If $a \neq 0$, then $a \cdot b = a \cdot c$ implies $b = c$.

Algebraic Expressions

An *algebraic expression* is a string of symbols used to represent an element in a carrier of an algebra. For example, 3, $8 \div x$, and $x^2 + y$ are algebraic expressions in high school algebra. But $x + y +$ is not an algebraic expression. The set of algebraic expressions is a language. The symbols in the alphabet are the operators and constants from the algebra together with variable names and grouping symbols, like parentheses and commas. The language of algebraic expressions over an algebra can be defined inductively as follows:

Algebraic Expressions

1. Constants and variables are algebraic expressions.
2. An operator applied to its arguments is an algebraic expression if the arguments are algebraic expressions.

For example, suppose x and y are variables and c is a constant. If g is a ternary operator, then the following five strings are algebraic expressions:

$$x,\ y,\ c,\ g(x, y, c),\ g(x, g(c, y, x), x).$$

Different algebraic expressions often mean the same thing. For example, the equation $2x = x + x$ makes sense to us because we look beyond the two strings $2x$ and $x + x$, which are not equal strings. Instead, we look at the possible values of the two expressions and conclude that they always have the same value, no matter what value x has. Two algebraic expressions are *equivalent* if they always evaluate to the same element in a carrier of the algebra. So the expressions $2x$ and $x + x$ are equivalent in high school algebra. We can make the idea of equivalence precise by giving an inductive definition. Assume that C is a carrier of an algebra.

Equivalent Algebraic Expressions

1. Any element in C is equivalent to itself.
2. Suppose E and E' are two algebraic expressions and x is a variable such that $E(x/b)$ and $E'(x/b)$ are equivalent for all elements b in C. Then E is equivalent to E'.

For example, the two expressions $(x + 2)^2$ and $x^2 + 4x + 4$ are equivalent in high school algebra. But $x + y$ is not equivalent to $5x$ because we can let $x = 1$ and $y = 2$, which makes $x + y = 3$ and $5x = 5$.

Describing an Algebra

The set of operators in an algebra is called the *signature* of the algebra. When describing an algebra, we need to decide which operators to put in the signature. For example, we may wish to list only the primitive operators (the constructors) that are used to build all other operators. On the other hand, we might want to list all the operators that we know about.

Let's look at a convenient way to denote an algebra. We'll list the carrier or carriers first, followed by a semicolon. The operators in the signature are listed next. For example, this notation is used to denote the following algebras.

$$\begin{array}{ll}
\langle \mathbb{N};\ \text{succ},\ 0\rangle & \langle \mathbb{N};\ +,\ \cdot,\ 0,\ 1\rangle \\
\langle \mathbb{N};\ +,\ 0\rangle & \langle \mathbb{Z};\ +,\ \cdot,\ -,\ 0,\ 1\rangle \\
\langle \mathbb{N};\ \cdot,\ 1\rangle & \langle \mathbb{Q};\ +,\ \cdot,\ -,\ \div,\ 0,\ 1\rangle \\
\langle \mathbb{N};\ \text{succ},\ +,\ 0\rangle & \langle \mathbb{R};\ +,\ \cdot,\ -,\ \div,\ 0,\ 1\rangle
\end{array}$$

The constants 0 and 1 are listed as operations to emphasize the fact that they have special properties, such as $x + 0 = x$ and $x \cdot 1 = x$.

It may also be convenient to use a picture to describe an algebra. The diagram in Figure 10.1 represents the algebra $\langle \mathbb{N};\ +,\ 0\rangle$.

The circle represents the carrier $\mathbb{N}$, of natural numbers. The two arrows coming out of $\mathbb{N}$ represent two arguments to the $+$ operator. The arrow from $+$ to $\mathbb{N}$ indicates that the result of $+$ is an element of $\mathbb{N}$. The fact that there are no arrows pointing at 0 means that 0 is a constant (an operator with no arguments), and the arrow from 0 to $\mathbb{N}$ means that 0 is an element of $\mathbb{N}$.

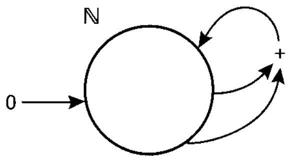


Figure 10.1 An algebra.

10.1.2 Concrete Versus Abstract

An algebra is called *concrete* if its carriers are specific sets of elements so that its operators are defined by rules applied to the carrier elements. High school algebra is a concrete algebra. In fact, all the examples that we have seen so far are concrete algebras.

An algebra is called *abstract* if it is not concrete. In other words, its carriers don't have any specific set interpretation. Thus its operators cannot be defined in terms of rules applied to the carrier elements because we don't have a description of them. Therefore the general properties of the operators in an abstract algebra must be given by axioms. An abstract algebra is a powerful description tool because it represents all the concrete algebras that satisfy its axioms.

So when we talk about an abstract algebra, we are really talking about all possible examples of the algebra. Is this a useful activity? Sure. Many times we are overwhelmed with important concepts, but we aren't given any tools to make sense of them. Abstraction can help to classify things and thus make sense of things that act in similar ways.

If an algebra is abstract, then we must be more explicit when trying to describe it. For example, suppose we write down the following algebra:

$$\langle S;\ s,\ a\rangle.$$

All we know at this point is that S is a carrier and there are two operators s and a. We don't even know the arity of s or a.

Suppose we're told that a is a constant of S and s is a unary operator on S. Now we know something, but not very much, about the algebra. We can use the operators s and a to construct the following algebraic expressions for elements of S:

$$a,\ s(a),\ s(s(a)),\ \ldots,\ s^n(a),\ \ldots.$$

This is the most we can say about the elements of S. There might be other elements in S, but we have no way of knowing it. The elements of S that we know about can be represented by all possible algebraic expressions made up from the operator symbols in the signature together with left and right parentheses.

example 10.1 Induction Algebra

An algebra $\langle S;\ s,\ a\rangle$ is called an *induction algebra* if s is a unary operator on S and a is a constant of S such that

$$S = \{a,\ s(a),\ s(s(a)),\ \ldots,\ s^n(a),\ \ldots\}.$$

The word "induction" is used because of the natural ordering on the carrier that can be used in inductive proofs.

The algebra $\langle \mathbb{N}; \text{succ}, 0\rangle$ is a concrete example of an induction algebra, where $\text{succ}(x) = x + 1$. The algebraic expressions for the elements are

$$0, \text{succ}(0), \text{succ}(\text{succ}(0)), \ldots, \text{succ}^n(0), \ldots,$$

where $\text{succ}(0) = 1$, $\text{succ}(\text{succ}(0)) = 2$, and so on. So every natural number is represented by one of the algebraic expressions.

There are many concrete examples of induction algebras. For example, let $A = \{2, 1, 0, -1, -2, -3, \ldots\}$. Then the algebra $\langle A; \text{pred}, 2\rangle$ is an induction algebra, where $\text{pred}(x) = x - 1$. The expressions for the elements are

$$2, \text{pred}(2), \text{pred}(\text{pred}(2)), \ldots, \text{pred}^n(2), \ldots,$$

where we have $\text{pred}(2) = 1$, $\text{pred}(\text{pred}(2)) = 0$, and so on. So every number in A is represented by one of the algebraic expressions.

end example

Axioms for Abstractions

Interesting things can happen when we add axioms to an abstract algebra. For example, the algebra $\langle S; s, a\rangle$ changes its character when we add the single axiom $s^6(x) = x$ for all $x \in S$. In this case we can say that the algebraic expressions define a finite set of elements, which can be represented by the following six expressions:

$$a, s(a), s^2(a), s^3(a), s^4(a), s^5(a).$$

A complete definition of an abstract algebra can be given by listing the carriers, operations, and axioms. For example, the abstract algebra that we've just been discussing can be defined as follows.

Carrier:	S
Operations:	$a \in S$
	$s : S \to S$
Axiom:	$s^6(x) = x$.

We'll always assume that the variable x is universally quantified over S.

example **10.2 A Finite Algebra**

The algebra $\langle \mathbb{N}_6; \text{succ}_6, 0\rangle$, where $\text{succ}_6(x) = (x + 1) \bmod 6$, is a concrete example of the abstract algebra $\langle S; s, a\rangle$ with axiom $s^6(x) = x$. To see this, observe that the algebraic expressions for the carrier elements are

$$0, \text{succ}_6(0), \text{succ}_6(\text{succ}_6(0)), \ldots, \text{succ}_6^n(x), \ldots.$$

But we have $\text{succ}_6(x) = (x + 1) \bmod 6$. So the preceding algebraic expressions evaluate to an infinite repetition of six numbers in $\mathbb{N}_6$.

$$0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5, \ldots.$$

In other words, we have $\text{succ}_6^6(x) = x$ for all $x \in \mathbb{N}_6$, which has the same form as the axiom in the abstract algebra.

end example

10.1.3 Working in Algebras

The goal of the next paragraphs is to get familiar with some elementary properties of algebraic operations. We'll see more examples of algebras and we'll observe whether they have any of the properties we've discussed.

Properties of the Operations

Let's look at some fundamental properties that may be associated with a binary operation. If $\circ$ is a binary operator on a set C, then an element $z \in C$ is called a *zero* for $\circ$ if the following condition holds:

$$z \circ x = x \circ z = z \quad \text{for all } x \in C.$$

For example, the number 0 is a zero for the multiply operation over the real numbers because $0 \cdot x = x \cdot 0 = 0$ for all real numbers x.

Continuing with the same binary operator $\circ$ and carrier C, we call an element $u \in C$ an *identity*, or *unit*, for $\circ$ if the following condition holds:

$$u \circ x = x \circ u = x \quad \text{for all } x \in C.$$

For example, the number 1 is an identity for the multiply operation over the real numbers because $1 \cdot x = x \cdot 1 = x$ for all numbers x. Similarly, the number 0 is an identity for the addition operation over real numbers because $0 + x = x + 0 = x$ for all numbers x.

Suppose u is an identity element for $\circ$, and $x \in C$. An element y in C is called an *inverse* of x if the following equation holds:

$$x \circ y = y \circ x = u.$$

For example, in the algebra $\langle \mathbb{Q}; \cdot, 1\rangle$ the number 1 is an identity element. We also know that if $x \neq 0$, then

$$x \cdot \frac{1}{x} = \frac{1}{x} \cdot x = 1.$$

In other words, all nonzero rational numbers have inverses.

Each of the following examples presents an algebra together with some observations about its operators.

example **10.3 Algebra of Sets**

Let S be a set. Then the power set of S is the carrier for an algebra described as follows:

$$\langle \text{power}(S); \cup, \cap, \varnothing, S\rangle.$$

Notice that if $A \in \text{power}(S)$, then $A \cup \varnothing = A$, and $A \cap S = A$. So $\varnothing$ is an identity for $\cup$, and S is an identity for $\cap$. Similarly, $A \cap \varnothing = \varnothing$, and $A \cup S = S$. Thus $\varnothing$ is a zero for $\cap$, and S is a zero for $\cup$. This algebra has many well-known properties. For example, $A \cup A = A$ and $A \cap A = A$ for any $A \in \text{power}(S)$. We also know that $\cap$ and $\cup$ are commutative and associative and that they distribute over each other.

end example

example **10.4 A Finite Algebra**

Let $\mathbb{N}_n$ denote the set $\{0, 1, \ldots, n - 1\}$, and let "max" be the function that returns the maximum of its two arguments. Consider the following algebra with carrier $\mathbb{N}_n$:

$$\langle \mathbb{N}_n; \max, 0, n - 1\rangle.$$

Notice that max is commutative and associative. Notice also that for any $x \in \mathbb{N}_n$ it follows that $\max(x, 0) = \max(0, x) = x$. So 0 is an identity for max. It's also easy to see that for any $x \in \mathbb{N}_n$,

$$\max(x, n - 1) = \max(n - 1, x) = n - 1.$$

So $n - 1$ is a zero for the operator max.

end example

example **10.5 An Algebra of Functions**

Let S be a set, and let F be the set of all functions of type $S \to S$. If we let $\circ$ denote the operation of composition of functions, then F is the carrier of an algebra $\langle F; \circ, \text{id}\rangle$. The function "id" denotes the identity function. In other words, we have the equation $\text{id} \circ f = f \circ \text{id} = f$ for all functions f in F. Therefore, id is an identity for $\circ$.

end example

Notice that we used the equality symbol "=" in the above examples without explicitly defining it as a relation. The first example uses equality of sets, the second uses equality of numbers, and the third uses equality of functions. In our discussions we will usually assume an implicit equality theory on each carrier of an algebra. But, as we have said before, equality relations are operations that may need to be implemented when needed as part of a programming activity.

$\circ$	a	b	c	d
a	a	b	c	d
b	b	c	d	a
c	c	d	a	b
d	d	a	b	c

Figure 10.2 Binary operation table.

Operation Tables

Any binary operation on a finite set can be represented by a table, called an *operation table.* For example, if $\circ$ is a binary operation on the set $\{a, b, c, d\}$, then the operation table for $\circ$ might look like Figure 10.2, where the elements of the set are used as row labels and column labels.

If x is a row label and y is a column label, then the element in the table at row x and column y represents the element $x \circ y$. For example, we have $c \circ d = b$.

We can often find out many things about a binary operation by observing its operation table. For example, notice in Figure 10.2 that the row labeled a and the column labeled a are copies of the row label and column label sequence $a\ b\ c\ d$. This tells us that a is an identity for $\circ$. It's also easy to see that $\circ$ is commutative and that each element has an inverse. Does $\circ$ have a zero? It's easy to see that the answer is no. Is $\circ$ associative? The answer is yes, but it's not very easy to check. We'll leave these problems as exercises.

It's also easy to see that there cannot be more than one identity for a binary operation. Can you see why from Figure 10.2? We'll prove the following general fact about identities for any binary operation.

Uniqueness of an Identity **(10.1)**

Any binary operation has at most one identity.

Proof: Let $\circ$ be a binary operation on a set S. To show that $\circ$ has at most one identity, we'll assume that u and e are identities for $\circ$. Then we'll show that $u = e$. Remember, since u and e are identities, we know that $u \circ x = x \circ u = x$ and $e \circ x = x \circ e = x$ for all x in S. Thus we have the following equality:

$$\begin{aligned} e &= e \circ u && (u \text{ is an identity for } \circ) \\ &= u && (e \text{ is an identity for } \circ). \quad \text{QED} \end{aligned}$$

Algebras with One Binary Operation

Some algebras are used so frequently that they have been given names. For example, any algebra of the form $\langle A; \circ \rangle$, where $\circ$ is a binary operation, is called a *groupoid.* If we know that the binary operation is associative, then the algebra is called a *semigroup.* If we know that the binary operation is associative and also has an identity, then the algebra is called a *monoid.* If we know that the binary operation is associative, has an identity, and each element has an inverse, then the algebra is called a *group.* So these words are used to denote certain properties of the binary operation. Here's a synopsis.

Groupoid:	$\circ$ is a binary operation.
Semigroup:	$\circ$ is an associative binary operation.
Monoid:	$\circ$ is an associative binary operation with an identity.
Group:	$\circ$ is an associative binary operation with an identity and every element has an inverse.

It's clear from the listing of properties that a group is a monoid, which is a semigroup, which is a groupoid. But things don't go the other way. For example, the algebra in Example 10.5 is a monoid but not a group, since not every function has an inverse.

We can have some fun with these names. For example, we can describe a group as a monoid with inverses, and we can describe a monoid as a semigroup with identity. When an algebra contains an operation that satisfies some special property beyond the axioms of the algebra, we often modify the name of the algebra with the name of the property. For example, the algebra $\langle \mathbb{Z};\ +,\ 0\rangle$ is a group. But we know that the operation + is commutative. Therefore, we can call the algebra a "commutative" group.

Now let's discuss a few elementary results. To get our feet wet, we'll prove the following simple property that holds in any monoid.

Uniqueness of Inverses **(10.2)**

In a monoid, if an element has an inverse, the inverse is unique.

Proof: Let $\langle M;\ \circ,\ u\rangle$ be a monoid. We will show that if an element x in M has an inverse, then the inverse is unique. In other words, if y and z are both inverses of x, then $y = z$. We can prove this result as follows:

$$\begin{aligned}
y &= y \circ u && (u \text{ is the identity for } \circ)\\
&= y \circ (x \circ z) && (z \text{ is an inverse of } x)\\
&= (y \circ x) \circ z && (\circ \text{ is associative})\\
&= u \circ z && (y \text{ is an inverse of } x)\\
&= z && (u \text{ is the identity for } \circ).\quad \text{QED}
\end{aligned}$$

If we have a group, then we know that every element has an inverse. Thus we can conclude from (10.2) that every element x in a group has a unique inverse, which is usually denoted by writing the symbol

$$x^{-1}.$$

example

10.6 Working with Groups

We can use the elementary properties of a group to obtain other properties. For example, let $\langle G;\ \circ,\ e\rangle$ be a group. This means that we know that $\circ$ is associative, e is an identity, and every element of G has an inverse. We'll prove two properties as examples. The first property is stated as follows:

$$\text{If } x \circ x = x \text{ holds for some element } x \in G, \text{ then } x = e. \tag{10.3}$$

Proof:
$$\begin{aligned} x &= x \circ e && (e \text{ is an identity for } \circ)\\ &= x \circ (x \circ x^{-1}) && (x^{-1} \text{ is the inverse of } x)\\ &= (x \circ x) \circ x^{-1} && (\circ \text{ is associative})\\ &= x \circ x^{-1} && (x \circ x = x \text{ is the hypothesis})\\ &= e && (x^{-1} \text{ is the inverse of } x).\ \text{QED} \end{aligned}$$

A second property of groups is cancellation on the left. This property can be stated as follows:

$$\text{If } x \circ y = x \circ z \text{ then } y = z \tag{10.4}$$

Proof:
$$\begin{aligned} y &= e \circ y && (e \text{ is an identity})\\ &= (x^{-1} \circ x) \circ y && (x^{-1} \text{ is the inverse of } x)\\ &= x^{-1} \circ (x \circ y) && (\circ \text{ is associative})\\ &= x^{-1} \circ (x \circ z) && (\text{hypothesis})\\ &= (x^{-1} \circ x) \circ z && (\circ \text{ is associative})\\ &= e \circ z && (x^{-1} \text{ is the inverse of } x)\\ &= z && (e \text{ is an identity}).\ \text{QED} \end{aligned}$$

end example

Algebras with Several Operations

A natural example of an algebra with two binary operations is the integers together with the usual operations of addition and multiplication. We can denote this algebra by the structure $\langle \mathbb{Z};\ +,\ \cdot,\ 0,\ 1\rangle$. This algebra is a concrete example of an algebra called a ring, which we'll now define. A *ring* is an algebra with the structure

$$\langle A;\ +,\ \cdot,\ 0,\ 1\rangle,$$

$+_5$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

$\cdot_5$	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Figure 10.3 Mod 5 addition and multiplication tables.

where $\langle A;\ +,\ 0\rangle$ is a commutative group, $\langle A;\ \cdot,\ 1\rangle$ is a monoid, and the operation $\cdot$ distributes over $+$ from the left and the right. This means that

$$a \cdot (b + c) = a \cdot b + a \cdot c \quad \text{and} \quad (b + c) \cdot a = b \cdot a + c \cdot a.$$

Check to see that $\langle \mathbb{Z};\ +,\ \cdot,\ 0,\ 1\rangle$ is indeed a ring.

If $\langle A;\ +,\ \cdot,\ 0,\ 1\rangle$ is a ring with the additional property that $\langle A - \{0\};\ \cdot,\ 1\rangle$ is a commutative group, then it's called a *field.* The ring $\langle \mathbb{Z};\ +,\ \cdot,\ 0,\ 1\rangle$ is not a field because, for example, 3 does not have an inverse for multiplication. On the other hand, if we replace $\mathbb{Z}$ by $\mathbb{Q}$, the rational numbers, then $\langle \mathbb{Q};\ +,\ \cdot,\ 0,\ 1\rangle$ is a field. For example, 3 has inverse 1/3 in $\mathbb{Q} - \{0\}$.

For another example of a field, let $\mathbb{N}_5 = \{0, 1, 2, 3, 4\}$ and let $+_5$ and $\cdot_5$ be addition mod 5 and multiplication mod 5, respectively. Then $\langle \mathbb{N}_5;\ +_5,\ \cdot_5,\ 0,\ 1\rangle$ is a field. Figure 10.3 shows the operation tables for $+_5$ and $\cdot_5$. We'll leave the verification of the field properties as an exercise.

The next examples show some algebras that might be familiar to you.

example 10.7 Polynomial Algebras

Let $\mathbb{R}[x]$ denote the set of all polynomials over x with real numbers as coefficients. It's a natural process to add and multiply two polynomials. So we have an algebra $\langle \mathbb{R}[x];\ +,\ \cdot,\ 0,\ 1\rangle$, where $+$ and $\cdot$ represent addition and multiplication of polynomials and 0 and 1 represent themselves. This algebra is a ring. Why isn't it a field?

end example

example 10.8 Matrix Algebras

Suppose we let $M_n(\mathbb{R})$ denote the set of all n by n matrices with elements in $\mathbb{R}$. We can add two matrices A and B by letting $A_{ij} + B_{ij}$ be the element in the ith row and jth column of the sum. We can multiply A and B by letting $\sum_{k=1}^{n} A_{ik}B_{kj}$ be the element in the ith row and jth column of the product. Thus we have an algebra $\langle M_n(\mathbb{R});\ +,\ \cdot,\ 0,\ 1\rangle$, where $+$ and $\cdot$ represent matrix addition and multiplication, 0 represents the matrix with all entries zero, and 1 represents the matrix with 1's along the main diagonal and 0's elsewhere. This algebra is a ring. Why isn't it a field?

end example

example
10.9 Vector Algebras

The algebra of n-dimensional vectors, with real numbers as components, can be described by listing two carriers $\mathbb{R}$ and $\mathbb{R}^n$. We can multiply a vector $(x_1, \ldots, x_n) \in \mathbb{R}^n$ by number $b \in \mathbb{R}$ to obtain a new vector by multiplying each component of the vector by b, obtaining

$$(bx_1, \ldots, bx_n).$$

If we let $\cdot$ denote this operation, then we have

$$b \cdot (x_1, \ldots, x_n) = (bx_1, \ldots, bx_n).$$

We can add vectors by adding corresponding components. For example,

$$(x_1, \ldots, x_n) + (y_1, \ldots, y_n) = (x_1 + y_1, \ldots, x_n + y_n).$$

Thus we have an algebra $\langle \mathbb{R}, \mathbb{R}^n; \cdot , + \rangle$ of n-dimensional vectors. Notice that the algebra has two carriers, $\mathbb{R}$ and $\mathbb{R}^n$. This is because they are both necessary to define the $\cdot$ operation, which has type $\mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$.

end example

example
10.10 Power Series Algebras

If we extend polynomials over x to allow infinitely many terms, then we obtain what are called *power series* (we also know them as generating functions). Letting $\mathbb{R}[[x]]$ denote the set of power series with real numbers as coefficients, we obtain the algebra $\langle \mathbb{R}[[x]]; +, \cdot, 0, 1\rangle$, where $+$ and $\cdot$ represent addition and multiplication of power series and 0 and 1 represent themselves. This algebra is a ring. Why isn't it a field?

end example

Exercises

Algebraic Properties

1. Let m and n be two integers with $m < n$. Let $A = \{m, m + 1, \ldots, n\}$, and let "min" be the function that returns the smaller of its two arguments. Does min have a zero? Identity? Inverses? If so, describe them.

2. Let $A = \{\text{true, false}\}$. For each of the following binary operations on A, answer the three questions: Does the operation have a zero? Does the operation have an identity? What about inverses?

   a. Conditional, $\rightarrow$.
   b. Conjunction, $\wedge$.
   c. Disjunction, $\vee$.

3. Given the algebra $\langle S; f, a\rangle$, where f is a unary operation and a is a constant of S and $f^5(x) = f^3(x)$ for all $x \in S$, find a finite set of algebraic expressions that will represent the distinct elements of S.

4. Given a binary operation on a finite set in table form, for each of the following parts, describe an easy way to detect whether the binary operation has the listed property.

   a. There is a zero.
   b. The operation is commutative.
   c. Inverses exist for each element of the set (assume that there is an identity).

5. Let $A = \{a, b, c, d\}$, and let $\circ$ be a binary operation on A. For each of the following problems, write down a table for $\circ$ that satisfies the given properties.

   a. a is an identity for $\circ$, but no other element of A has an inverse.
   b. a is an identity for $\circ$, and every element of A has an inverse.
   c. a is a zero for $\circ$, and $\circ$ is not associative.
   d. a is an identity, and exactly two elements have inverses.
   e. a is an identity for $\circ$, and $\circ$ is commutative but not associative.

6. Let $A = \{a, b\}$. For each of the following problems, find an operation table satisfying the given condition for a binary operation $\circ$ on A.

   a. $\langle A; \circ\rangle$ is a group.
   b. $\langle A; \circ\rangle$ is a monoid but not a group.
   c. $\langle A; \circ\rangle$ is a semigroup but not a monoid.
   d. $\langle A; \circ\rangle$ is a groupoid but not a semigroup.

7. Write an algorithm to check a binary operation table for associativity.

Challenges

8. Given the algebra $\langle S; f, g, a\rangle$, where f and g are unary operations and a is a constant of S, suppose that $f(f(x)) = g(x)$ and $g(g(x)) = x$ for all $x \in S$.

a. Show that $f(g(x)) = g(f(x))$ for all $x \in S$.
b. Show that $f(f(f(f(x)))) = x$ for all $x \in S$.
c. Find a finite set of algebraic expressions to represent the distinct elements of S.

9. Prove each of the following facts about a group $\langle G;\ \circ,\ e\rangle$.

   a. Cancellation on the right: If $y \circ x = z \circ x$, then $y = z$.
   b. The inverse of $x \circ y$ is $y^{-1} \circ x^{-1}$. In other words, $(x \circ y)^{-1} = y^{-1} \circ x^{-1}$.

10. Let $\mathbb{N}_5 = \{0, 1, 2, 3, 4\}$, and let $+_5$ and $\cdot_5$ be the two operations of addition mod 5 and multiplication mod 5, respectively. Show that $\langle \mathbb{N}_5;\ +_5,\ \cdot_5,\ 0,\ 1\rangle$ is a field.

10.2 Boolean Algebra

Do the techniques of set theory and the techniques of logic have anything in common? Let's do an example to see that the answer is yes. When working with sets, we know that the following equation holds for all sets A, B, and C:

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C).$$

When working with propositions, we know that the following equivalence holds for all propositions A, B, and C:

$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C).$$

Certainly these two examples have a similar pattern. As we'll see shortly, sets and logic have a lot in common. They can both be described as concrete examples of a Boolean algebra. The name "Boolean" comes from the mathematician George Boole (1815–1864), who studied relationships between set theory and logic. Let's get to the definition.

Definition of Boolean Algebra

A *Boolean algebra* is an algebra with the structure $\langle B;\ +,\ \cdot,\ ^{-},\ 0,\ 1\rangle$, where the following properties hold.

Defining Properties of a Boolean Algebra

1. $\langle B;\ +,\ 0\rangle$ and $\langle B;\ \cdot,\ 1\rangle$ are commutative monoids. In other words, the following properties hold for all $x,\ y,\ z \in B$:

$$\begin{aligned} &(x+y)+z = x+(y+z), && (x \cdot y) \cdot z = x \cdot (y \cdot z), \\ &x+y = y+x, && x \cdot y = y \cdot x, \\ &x+0 = x, && x \cdot 1 = x. \end{aligned}$$

2. The operations $+$ and $\cdot$ distribute over each other. In other words, the following properties hold for all $x,\ y,\ z \in B$:

$$x \cdot (y + z) = (x \cdot y) + (x \cdot z) \quad \text{and} \quad x + (y \cdot z) = (x + y) \cdot (x + z).$$

3. $x + \overline{x} = 1$ and $x \cdot \overline{x} = 0$ for all elements $x \in B$. The element $\overline{x}$ is called the *complement* of x or the *negation* of x.

We often drop the dot and write xy in place of $x \cdot y$. We'll also reduce the need for parentheses by agreeing to the following precedence hierarchy:

$$\begin{array}{cl} \overline{} & \text{highest (do it first),} \\ \cdot & \\ + & \text{lowest (do it last).} \end{array}$$

For example, the expression $a + b\overline{c}$ means the same thing as $(a + (b\,(\overline{c})))$.

example

10.11 Sets

Suppose B = power(S) for some set S. Then B is the carrier of a Boolean algebra if we let union and intersection act as the operations $+$ and $\cdot$, let X' be the complement of X, let $\varnothing$ act as 0, and let S act as 1. For example, the two properties in part 3 of the definition are represented by the following equations, where X is any subset of S:

$$X \cup X' = S \quad \text{and} \quad X \cap X' = \varnothing.$$

end example

example

10.12 Logic

Suppose we let B be the set of all propositional wffs of the propositional calculus. Then B is the carrier of a Boolean algebra if we let disjunction and conjunction act as the operations $+$ and $\cdot$, let $\neg\ X$ be the complement of X, let false act as 0, let true act as 1, and let logical equivalence act as equality. For example, the two properties in part 3 of the definition are represented by the following equivalences, where X is any proposition:

$$X \vee \neg X \equiv \text{true} \quad \text{and} \quad X \wedge \neg X \equiv \text{false}.$$

We can also obtain a very simple Boolean algebra by using just the carrier {false, true} together with the operations $\vee$, $\wedge$, and $\neg$.

end example

example

10.13 Divisors

Let n be a product of distinct prime numbers. For example, n could be 30 because $30 = 2{\cdot}3{\cdot}5$, but n cannot be 12 because $12 = 2{\cdot}2{\cdot}3$, which is not a product of distinct primes. Let B_n be the set of positive divisors of n. Then B_n is the carrier of a Boolean algebra if we let "least common multiple" and "greatest common divisor" be the operations $+$ and $\cdot$, respectively. Let n/x be the complement of x, let 1 act as the zero, and let n act as the one.

With these definitions, all the properties of a Boolean algebra are satisfied. For example, the two properties in part 3 of the definition are represented by the following equations, where $x \in B_n$:

$$\text{lcm}(x,\ n/x) = n \quad \text{and} \quad \text{gcd}(x,\ n/x) = 1.$$

For an example, let $n = 10 = 2{\cdot}5$. Then $B_{10} = \{1, 2, 5, 10\}$, 1 is the zero, and 10 is the one. Thus, for example, the complement of 2 is 5, lcm(2, 5) = 10 (the one), and gcd(2, 5) = 1 (the zero).

Notice what happens if we let $n = 12$. We get $B_{12} = \{1, 2, 3, 4, 6, 12\}$. The reason B_{12} does not yield a Boolean algebra with our definition is because 2 and its complement 6 don't satisfy the properties in part 3 of the definition. Notice that lcm(2, 6) = 6, which is not the one, and gcd(2, 6) = 2, which is not the zero.

end example

10.2.1 Simplifying Boolean Expressions

A fundamental problem of Boolean algebra, with applications to such areas as logic design and theorem-proving systems, is to simplify Boolean expressions so that they contain a small number of operations. Let's see how the axioms of Boolean algebra can help us obtain some useful simplification properties.

For example, in the Boolean algebra of propositions we have $P \wedge P \equiv P$ for any proposition P. Similarly, in the Boolean algebra of sets we have $S \cap S = S$ for any set S. Can we generalize these properties to all Boolean algebras? In other words, can we say $bb = b$ for every element b in the carrier of a Boolean algebra? The answer is yes. Let's prove it with equational reasoning. Be sure you can provide a reason for each step of the following proof:

$$b = b \cdot 1 = b \cdot \left(b + \overline{b}\right) = b \cdot b + b \cdot \overline{b} = b \cdot b + 0 = b \cdot b.$$

A related statement is $b + b = b$ for all elements b. Can you provide the proof? We'll state these two properties for the record.

Idempotent Properties **(10.5)**

$$b \cdot b = b \quad \text{and} \quad b + b = b.$$

A nice property of Boolean algebras is that results come in pairs. This is because the axioms come in pairs. In other words, $\langle B;\ +,\ 0\rangle$ and $\langle B;\ \cdot,\ 1\rangle$ are both commutative monoids; $+$ and $\cdot$ distribute over each other; and $b + \overline{b} = 1$ and $b \cdot \overline{b} = 0$ for all elements $b \in B$. The *duality principle* states that whenever a result A is true for a Boolean algebra, then a dual result A' is also true, where A' is obtained from the A by simultaneously replacing all occurrences of $\cdot$ by $+$, all occurrences of $+$ by $\cdot$, all occurrences of 1 by 0, and all occurrences of 0 by 1. A proof for the result A' can be obtained by making these same changes in the proof of A.

There are lots of properties that we can discover. For example, if S is a set, then $\varnothing \cap A = \varnothing$ for any subset A of S. This is an instance of a general property that holds for any Boolean algebra: $0 \cdot b = 0$ for every element b. This follows readily from (10.5) as follows:

$$0 \cdot b = \left(\overline{b} \cdot b\right) \cdot b = \overline{b} \cdot (b \cdot b) = \overline{b} \cdot b = 0.$$

Again, there is a dual result: $1 + b = 1$ for every b. Can you prove this result? We'll also state these two properties for the record:

Zero and One Properties **(10.6)**

$$0 \cdot b = 0 \quad \text{and} \quad 1 + b = 1.$$

Let's do an example to see how we can put our new knowledge to use in simplifying a Boolean expression. Suppose the function f is defined over a Boolean algebra by

$$f(x, y, z) = x + yz + z\overline{x}y + \overline{y}xz.$$

To evaluate f, we need to perform three $+$ operations, five $\cdot$ operations, and two $^{-}$ operations. Can we do any better? Sure we can. We can simplify the expression for $f(x, y, z)$ as follows—make sure you can state a reason for each line:

$$\begin{aligned} f(x,y,z) &= x + yz + z\overline{x}y + \overline{y}xz \\ &= x + yz(1+\overline{x}) + \overline{y}xz \\ &= x + yz1 + \overline{y}xz \\ &= x(1+\overline{y}z) + yz \\ &= x1 + yz \\ &= x + yz. \end{aligned}$$

So f can be evaluated with only one $+$ operation and one $\cdot$ operation.

To simplify Boolean expressions, it's important to have a good knowledge of Boolean algebra together with some luck and ingenuity. We'll give a few more general properties that are very useful simplification tools. The following properties can be used to simplify an expression by reducing the number of operations by two. We'll leave the proofs as exercises.

Absorption Laws **(10.7)**

$$\begin{aligned} a + ab &= a \quad \text{and} \quad a(a+b) = a. \\ a + \overline{a}b &= a + b \quad \text{and} \quad a(\overline{a}+b) = ab. \end{aligned}$$

In a Boolean algebra, complements are unique in the following sense: If an element acts like a complement of some element, then it is in fact the only complement of the element. Using symbols, we can state the result as follows:

Uniqueness of the Complement **(10.8)**

If $a + b = 1$ and $ab = 0$, then $b = \overline{a}$.

Proof: To prove this statement, we write the following equations:

$$\begin{aligned} b &= b1 \\ &= b(a+\overline{a}) \\ &= ba + b\overline{a} \\ &= 0 + b\overline{a} && \text{(since } ab = 0\text{)} \\ &= a\overline{a} + b\overline{a} \\ &= (a+b)\overline{a} \\ &= 1\overline{a} && \text{(since } a + b = 1\text{)} \\ &= \overline{a} && \text{QED.} \end{aligned}$$

Complements are quite useful in Boolean algebra. As a consequence of the uniqueness of complements (10.8), we have the following property:

Involution Law **(10.9)**

$$\bar{\bar{a}} = a.$$

Proof: Notice that $\bar{a} + a = 1$ and $\bar{a}a = 0$. Therefore, a acts like the complement of $\bar{a}$. Thus a is indeed equal to the complement of $\bar{a}$. That is, $a = \bar{\bar{a}}$. QED.

Recall from the propositional calculus that we have the following logical equivalence $\neg\ (p \wedge q) \equiv \neg\ p \vee \neg\ q$. This is an example of one of De Morgan's laws, which have the following forms in Boolean algebra.

De Morgan's Laws **(10.10)**

$$\overline{a+b} = \bar{a}\bar{b} \quad \text{and} \quad \overline{ab} = \bar{a} + \bar{b}.$$

Proof: We'll prove the first of the two laws and leave the second as an exercise. We'll use (10.8) to show that $\overline{a+b} = \bar{a}\bar{b}$. In other words, we'll show that $\bar{a}\bar{b}$ acts like the complement of $a + b$. Then we'll use (10.8) to conclude the result. First, we'll show that $(a + b) + \bar{a}\bar{b} = 1$ as follows:

$$\begin{aligned}(a+b)+\bar{a}\bar{b} &= \left(a+b+\bar{a}\right)\left(a+b+\bar{b}\right)\\ &= \left(a+\bar{a}+b\right)\left(a+b+\bar{b}\right)\\ &= (1+b)(a+1)\\ &= 1\cdot 1\\ &= 1.\end{aligned}$$

Next we'll show that $(a+b)\cdot\bar{a}\bar{b} = 0$ as follows:

$$\begin{aligned}(a+b)\cdot\bar{a}\bar{b} &= a\bar{a}\bar{b} + b\bar{a}\bar{b}\\ &= a\bar{a}\bar{b} + \bar{a}b\bar{b}\\ &= 0\bar{b} + \bar{a}0\\ &= 0+0\\ &= 0.\end{aligned}$$

Thus $\bar{a}\bar{b}$ acts like a complement of $a + b$. So we can apply (10.8) to conclude that $\bar{a}\bar{b}$ is the complement of $a + b$. In other words, $\overline{a+b} = \bar{a}\bar{b}$. QED.

In the propositional calculus we can find a disjunctive normal form (DNF) and a conjunctive normal form (CNF) for any wff. These ideas carry over to any Boolean algebra, where + corresponds to disjunction and · corresponds to conjunction. So we can make the following statement:

Any Boolean expression has a DNF and a CNF.

example **10.14 DNF and CNF Constructions**

We'll construct a DNF and CNF for the expression

$$\overline{a+b}+c.$$

The following transformations do the job:

$$\begin{aligned} \overline{a+b}+c &= \bar{a}\bar{b}+c && \text{(DNF)} \\ &= (\bar{a}+c)(\bar{b}+c) && \text{(CNF).} \end{aligned}$$

end example

10.2.2 Digital Circuits

Now let's see what Boolean algebra has to do with digital circuits. A *digital circuit* (also called a *logic circuit*) is an electronic representation of a function whose input values are either high or low voltages and whose output value is either a high or low voltage. Digital circuits are used to represent and process information in digital computers. The high- and low-voltage values are normally represented by the two digits 1 and 0.

The basic electronic components used to build digital circuits are called *gates*. The three basic "logic" gates are the AND gate, the OR gate, and the NOT gate. These gates work just like the corresponding logical operations, where 1 means true and 0 means false. So we can represent digital circuits as Boolean expressions with values in the Boolean algebra whose carrier is {0, 1}, where 0 means false, 1 means true, and the operations +, ·, and $^-$ stand for $\vee$, $\wedge$, and $\neg$, respectively.

The three logic gates are represented graphically as shown in Figure 10.4, where the inputs are on the left and the outputs are on the right.

Arithmetic Circuits

These gates can be combined in various ways to form digital circuits to do all basic arithmetic operations. For example, suppose we want to add two binary digits x and y. The first thing to notice is that the result has a summand digit and a carry digit. We'll consider two functions, "carry" and "summand." Let's look at the carry function first. Notice that carry(x, y) = 1 if and only if $x = 1$ and $y = 1$. Thus we can define the carry function as follows:

$$\text{carry}(x,\ y) = xy.$$

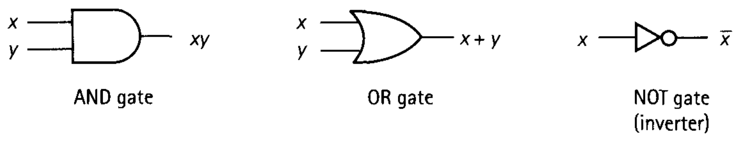


Figure 10.4 Logical gates.

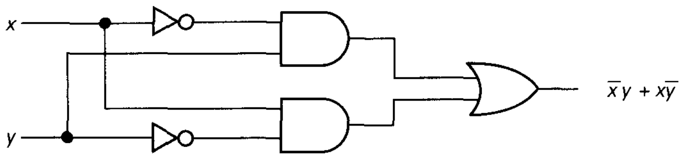


Figure 10.5 Summand circuit.

A circuit to implement the carry function consists of the simple AND gate shown in Figure 10.4.

Now let's look at the summand. It's clear that summand(x, y) = 1 if and only if either $x = 0$ and $y = 1$ or $x = 1$ and $y = 0$. Thus we can define the summand function as follows:

$$\text{summand}\,(x, y) = \overline{x}y + x\overline{y}.$$

A circuit to implement the summand function is shown in Figure 10.5.

We can combine the two circuits for the carry and the summand into one circuit that gives both outputs. The circuit is shown in Figure 10.6. Such a circuit is called a *half-adder*, and it's a fundamental building block in all arithmetic circuits.

example 10.15 A Simple Half-Adder

Let's see whether we can simplify the circuit for a half-adder. The preceding circuit for a half-adder has six gates. But we can do better. First, notice that the expression for the summand, $\overline{x}y + x\overline{y}$ has five operations: two negations, two

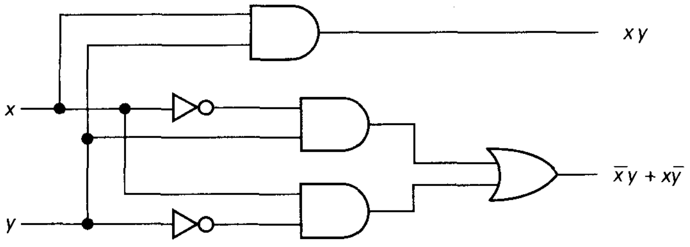


Figure 10.6 Half-adder circuit.

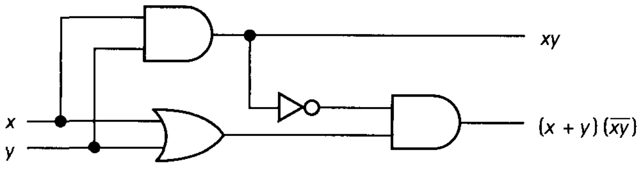


Figure 10.7 Simpler half-adder circuit.

conjunctions, and one disjunction. Let's rewrite it as follows (be sure to fill in the reasons for each step).

$$\begin{aligned}\overline{x}y + x\overline{y} &= (\overline{x}y + x)(\overline{x}y + \overline{y}) \\ &= (x + y)(\overline{x} + \overline{y}) \\ &= (x + y)\overline{xy}.\end{aligned}$$

This latter expression has four operations: two conjunctions, one disjunction, and one negation. Also, note that the expression xy is computed before the negation is applied. Therefore, we can also use this expression for the carry. So we have a simpler version of the half-adder, as shown in Figure 10.7.

end example

Constructing a Full Adder

We'll describe a circuit to add two binary numbers. For example, supppose we want to add the two binary numbers 1 0 1 1 and 1 1 1 0. The school method can be pictured as follows:

$$\begin{array}{ccccc} & 1 & 1 & 0 & \leftarrow \text{(carry bits)} \\ & 1 & 0 & 1 & 1 \\ & 1 & 1 & 1 & 0 \\ \hline 1 & 1 & 0 & 0 & 1 \end{array}$$

So if we want to add two binary numbers, then we can start by using a half-adder on the two rightmost digits of each number. After that, we must be able to handle the addition of three binary digits: two binary digits and a carry from the preceding addition. A digital circuit to accomplish this latter feat is called a *full adder.*

We can build a full adder by using half-adders as components. Let's see how it goes. First, to get the big picture, we will denote a half-adder by a box with two input lines and two output lines, as shown in Figure 10.8.

To get an idea about the kind of circuit we need, let's look at a table of values for the outputs sum and carry. Figure 10.9 shows the values of sum and carry that are obtained by adding three binary digits.

Let's use Figure 10.9 to find DNFs for the sum and carry functions in terms of x, y, and z. Notice that the value of the sum is 1 in four places, on lines 2,

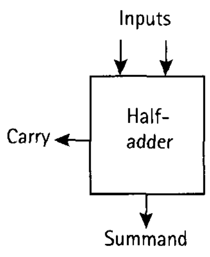


Figure 10.8 Half adder.

x	*y*	*z*	*Sum*	*Carry*
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 10.9 Adding three binary digits.

3, 5, and 8 of the table. So the DNF for the sum function will consist of the disjunction of four terms, with each conjunction constructed from the values of x, y, and z on the four lines. Similarly, the value of the carry is 1 in four places, on lines 4, 6, 7, and 8. So the DNF for the carry function depends on conjunctions that depend on these latter four lines. We obtain the following forms for sum and carry.

$$\begin{aligned}\operatorname{sum}(x,y,z) &= \overline{x}\,\overline{y}z + \overline{x}y\overline{z} + x\overline{y}\,\overline{z} + xyz \\ &= \overline{x}\left(\overline{y}z + y\overline{z}\right) + x\left(\overline{y}\,\overline{z} + yz\right).\end{aligned}$$

$$\begin{aligned}\operatorname{carry}(x,y,z) &= \overline{x}yz + x\overline{y}z + xy\overline{z} + xyz \\ &= yz + x\left(\overline{y}z + y\overline{z}\right).\end{aligned}$$

At this point we could build a circuit for sum and carry. But let's study the expressions that we obtained. Notice first that the expression

$$\overline{y}z + y\overline{z}$$

occurs in both the sum formula and the carry formula. Recall also that this is the expression for the summand output of the half-adder. It can be shown that the expression $\overline{y}\,\overline{z} + yz$, in the sum function, is equal to the negation of the

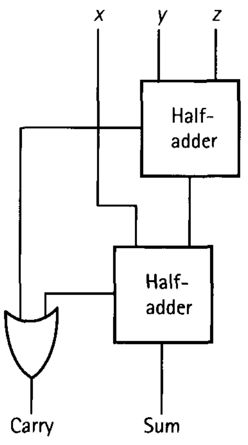


Figure 10.10 Full adder.

expression $\overline{y}z + y\overline{z}$ (the proof is left as an exercise). In other words, if we let $e = \overline{y}z + y\overline{z}$, then we can write the sum in the following form:

$$\text{sum}\,(x, y, z) = \overline{x}e + x\overline{e}.$$

This shows us that sum(x, y, z) is just the summand output of a half-adder. So we can let y and z be inputs to a half-adder and then feed the summand output along with x into another half-adder to obtain the desired sum(x, y, z). Before we draw the diagram, we need to look at the carry function. We have written carry in the following form:

$$\text{carry}\,(x, y, z) = yz + x\,(\overline{y}z + y\overline{z})\,.$$

Notice that the term yz is the carry output of a half-adder with input values y and z. Further, the term $x\,(\overline{y}z + y\overline{z})$ is the carry output of a half-adder with inputs x and $\overline{y}z + y\overline{z}$, where $\overline{y}z + y\overline{z}$ is the output of the half-adder with inputs y and z. So we can draw a picture of the circuit for a full adder as shown in Figure 10.10.

Minimization

As we've seen in the previous two examples, we can get simpler digital circuits if we spend some time simplifying the corresponding Boolean expressions. Often a digital circuit must be built with the minimum number of components, where the components correspond to DNFs or CNFs.

This brings up the question of finding a *minimal* DNF for a Boolean expression. Here the word "minimal" is usually defined to mean the fewest number of fundamental conjunctions in a DNF, and if two DNFs have the same number of fundamental conjunctions, then the one with the fewest literals is minimal.

The term *minimal* CNF is defined analogously. It's not always easy to find a minimal DNF for a Boolean expression.

example 10.16 A Minimal DNF

The Boolean expression $yz + yx$ is a minimal DNF for the expression

$$\overline{x}yz + xyz + xy\overline{z}.$$

We can show that these two expressions are equivalent as follows:

$$\begin{aligned}\overline{x}yz + xyz + xy\overline{z} &= (\overline{x} + x)\,yz + xy\overline{z}\\ &= yz + xy\overline{z}\\ &= y\,(z + x\overline{z})\\ &= y\,(z + x)\\ &= yz + yx.\end{aligned}$$

But it takes some work to see that $yz + xy$ is a minimal DNF. First we need to argue that there is no equivalent DNF with just a single fundamental conjunction. Then we need to argue that there is no equivalent DNF with two fundamental conjunctions with fewer than four literals.

end example

There are formal methods that can be applied to the problem of finding minimal DNFs and minimal CNFs. We'll leave them to more specialized texts.

Exercises

The Axioms

1. Let S be a set and B = power(S). Suppose someone claims that B is a Boolean algebra with the following definitions for the operators.

$+$	is union,
$\cdot$	is difference,
$-$	is complement with respect to S,
0	is S,
1	is $\varnothing$.

Is the result a Boolean algebra? Why or why not?

Boolean Expressions

2. Prove each of the following four absorption laws (10.7).

   a. $x + xy = x$. b. $x(x + y) = x$.

   c. $x + \overline{x}y = x + y$. d. $x(\overline{x} + y) = xy$.

3. Let $e = \overline{y}z + y\overline{z}$. Prove that $\overline{e} = \overline{y}\,\overline{z} + yz$.

4. Use Boolean algebra properties to prove each of the following equalities.

   a. $\overline{x} + \overline{y} + xyz = \overline{x} + \overline{y} + z$.

   b. $\overline{x} + \overline{y} + xy\overline{z} = \overline{x} + \overline{y} + \overline{xyz}$.

5. Simplify each of the following Boolean expressions.

   a. $x + \overline{x}y$ b. $x\overline{y}\,\overline{x} + xy\overline{x}$.

   c. $\overline{x}\,\overline{y}z + x\overline{y}\,\overline{z} + x\overline{y}z$. d. $xy + x\overline{y} + \overline{x}y$.

   e. $x(y + \overline{y}z) + \overline{y}z + yz$. f. $x + yz + \overline{x}y + \overline{y}xz$.

   g. $x + yz + \overline{x}y + \overline{y}xz$. h. $x + y(x + y)$.

   i. $(x + y) + \overline{x}\,\overline{y}$. j. $(x + y)(\overline{y} + x)(\overline{x} + y)$.

   k. $xy + x + y$. l. $(x + y)xy$.

6. For each part of Exercise 2, draw two logic circuits. One circuit should implement the expression on the left side of the equality. The other circuit should implement the expression on the right side of the equality. Each circuit should use the same number of gates as there are operations in the expression.

7. Write down the dual of each of the following Boolean expressions.

   a. $x + 1$. b. $x(y + z)$. c. $xy + xz$.

   d. $xy + z$. e. $y + \overline{x}z$. f. $z\overline{y}\,\overline{x} + xz\overline{x}$.

8. Show that, in a Boolean algebra, $1 + b = 1$ for every element b.

9. Show that, in a Boolean algebra, $\overline{a + b} = \overline{a} + \overline{b}$ for all elements a and b.

Challenges

10. Let B be the carrier of a Boolean algebra. Suppose B is a finite set, and suppose $0 \neq 1$. Show that the cardinality of B is an even number.

11. A Boolean algebra can be made into a partially ordered set by letting $a \preceq b$ mean $a = ab$.

    a. Show that $\preceq$ is reflexive, antisymmetric, and transitive.

    b. Show that $a \preceq b$ if and only if $b = a + b$.

12. A Boolean algebra, when considered as a poset—as in Exercise 11—is also a lattice. Prove that glb(a, b) = ab and lub(a, b) = $a + b$.

13. In Example 10.3 we considered the set B_n of positive divisors of n together with the operations of lcm, gcd, n/x, where n is one and 1 is zero. Prove that this algebra is not Boolean if a prime p occurs more than once as a factor of n. *Hint:* Consider the complement of p.

10.3 Abstract Data Types as Algebras

Programming problems involve data objects that are processed by a computer. To process data objects, we need operations to act on them. So algebra enters the programming picture. In computer science, an *abstract data type* consists of one or more sets of data objects together with one or more operations on the sets and some axioms to describe the operations. In other words, an abstract data type is an algebra. There is, however, a restriction on the carriers of abstract data types. A carrier must be able to be constructed in some way that will allow the data objects and the operations to be implemented on a computer.

Programming languages normally contain some built-in abstract data types. But it's not possible for a programming language to contain all possible ways to represent and operate on data objects. Therefore, programmers must often design and implement new abstract data types. The axioms of an abstract data type can be used by a programmer to check whether an implementation is correct. In other words, the implemented operations can be checked to see whether they satisfy the axioms.

An abstract data type allows us to program with its data objects and operations without having to worry about implementation details. For example, suppose we need to create an abstract data type for processing polynomials. We might agree to use the expression add(p, q) to represent the sum of two polynomials p and q. To implement the abstract data type, we might represent a polynomial as an array of coefficients and then implement the add operation by adding corresponding array components. Of course, there are other interesting and useful ways to represent polynomials and their addition. But no matter what implementation is used, the statement add(p, q) always means the same thing. So we've abstracted away the implementation details.

In this section we'll introduce some of the basic abstract data types of computer science.

10.3.1 Natural Numbers

In Chapter 3 we discussed the problem of trying to describe the natural numbers to a robot. Let's revisit the problem by trying to describe the natural numbers to ourselves from an algebraic point of view. We can start by trying out the following inductive definition:

1. $0 \in \mathbb{N}$.
2. There is a function $s : \mathbb{N} \to \mathbb{N}$ called "successor" with the following property: If $x \in \mathbb{N}$, then $s(x) \in \mathbb{N}$.

Does this inductive definition adequately describe the natural numbers? It depends on what we mean by "successor." For example, if $s(0) = 0$, then the set $\{0\}$ satisfies the definition. So the property $s(0) = 0$ must be ruled out. Let's try the additional axiom:

3. $s(x) \neq 0$ for all $x \in \mathbb{N}$.

Now $\{0\}$ doesn't satisfy the three axioms. But if we assume that $s(s(0)) = s(0)$, then the set $\{0, s(0)\}$ satisfies them. The problem here is that s sends two elements to the same place. We can eliminate this problem if we require s to be injective (one to one):

4. If $s(x) = s(y)$, then $x = y$.

This gives us the set of natural numbers in the form $0, s(0), s(s(0)), \ldots,$ where we set $s(0) = 1$, $s(s(0)) = 2$, and so on.

Historically, the first description of the natural numbers using axioms 1–4 was given by Peano. He also included a fifth axiom to describe the principle of mathematical induction:

5. If $q(x)$ is a property of x such that: $q(0)$ is true, and $q(x)$ implies $q(s(x))$; then $q(x)$ is true for all $x \in \mathbb{N}$.

Let's stop for a minute to write down an algebraic description of the natural numbers in terms of the first four rules:

Carrier:	$\mathbb{N}$.
Operations:	$0 \in \mathbb{N}$,
	$s : \mathbb{N} \to \mathbb{N}$.
Axioms:	$s(x) \neq 0$,
	If $s(x) = s(y)$, then $x = y$.

An algebra is useful as an abstract data type if we can define useful operations on the type in terms of its primitive operations. For example, can we define addition of natural numbers in this algebra? Sure. We can define the "plus" operation using only the successor operation as follows:

$$\begin{aligned} &\text{plus}(0, y) = y, \\ &\text{plus}(s(x), y) = s(\text{plus}(x, y)). \end{aligned}$$

For example, plus(2, 1) is computed by first writing $2 = s(s(0))$ and $1 = s(0)$. Then we can apply the definition recursively as follows:

$$\begin{aligned}\text{plus}(2,1) &= \text{plus}(s(s(0)), s(0))\\ &= s(\text{plus}(s(0), s(0)))\\ &= s(s(\text{plus}(0, s(0))))\\ &= s(s(s(0)))\\ &= 3.\end{aligned}$$

An alternative definition for the plus operation can be given as

$$\begin{aligned}&\text{plus}(0,y) = y,\\ &\text{plus}(s(x),y) = \text{plus}(x, s(y)).\end{aligned}$$

For example, using this definition, we can evaluate plus(2, 2) as follows:

$$\text{plus}(2,2) = \text{plus}(1,3) = \text{plus}(0,4) = 4.$$

Now that we have the plus operation, we can use it to define the multiplication operation as follows:

$$\begin{aligned}&\text{mult}(0,y) = 0,\\ &\text{mult}(s(x),y) = \text{plus}(\text{mult}(x,y),\ y).\end{aligned}$$

For example, we'll evaluate mult(3, 4) as follows—assuming that plus does its job properly:

$$\begin{aligned}\text{mult}(3,4) &= \text{plus}(\text{mult}(2,4),4)\\ &= \text{plus}(\text{plus}(\text{mult}(1,4),4),4)\\ &= \text{plus}(\text{plus}(\text{plus}(\text{mult}(0,4),4),4),4)\\ &= \text{plus}(\text{plus}(\text{plus}(0,4),4),4)\\ &= \text{plus}(\text{plus}(4,4),4)\\ &= \text{plus}(8,4) = 12.\end{aligned}$$

Let's see whether we can write the definitions for plus and mult in if-then-else form. To do so, we need the idea of a predecessor. Letting $p(x)$ denote the "predecessor" of x, we can write the definition of plus in either of the following ways:

$$\text{plus}(x,y) = \text{if } x = 0 \text{ then } y \text{ else } s(\text{plus}(p(x),y))$$

or

$$\text{plus}(x,y) = \text{if } x = 0 \text{ then } y \text{ else plus}(p(x), s(y)).$$

We'll leave it as an exercise to prove that these two definitions are equivalent. We can write the definition of mult as follows:

$$\text{mult}(x, y) = \text{if } x = 0 \text{ then } 0 \text{ else plus}(\text{mult}(p(x), y), y).$$

We can define the predecessor operation in terms of successor using the equation $p(s(x)) = x$. Since we're dealing only with natural numbers, we should either make $p(0)$ undefined or else define it so that it won't cause trouble. The usual definition is to say that $p(0) = 0$. It's interesting to note that we can't write an if-then-else definition for the predecessor using only the successor operation. So in some sense, the predecessor is a primitive operation too. So we'll add the definition of p to our algebra. We also need a test for zero to handle the test "$x = 0$" that occurs in if-then-else definitions.

To describe the algebra that includes these notions, we'll need another carrier to contain the true and false results that are returned by the test for zero. Letting Boolean = {true, false} and replacing s and p by the more descriptive names "succ" and "pred," we obtain the following algebra to represent the abstract data type of natural numbers:

Abstract Data Type of Natural Numbers **(10.11)**

Carriers: $\mathbb{N}$, Boolean.

Operations: $0 \in \mathbb{N}$,
isZero : $\mathbb{N} \to$ Boolean,
succ : $\mathbb{N} \to \mathbb{N}$,
pred : $\mathbb{N} \to \mathbb{N}$.

Axioms: isZero(0) = true,
isZero(succ(x)) = false,
pred(0) = 0,
pred(succ(x)) = x.

Notice that we've made some replacements. The old axiom succ(x) $\neq 0$ has been replaced by the new axiom, isZero(succ(x)) = false, which expresses the same idea. Also, the old axiom "If succ(x) = succ(y), then $x = y$" has been replaced by the new axiom pred(succ(x)) = x. To see this, notice that succ(x) = succ(y) implies that pred(succ(x)) = pred(succ(y)). Therefore, we can conclude that $x = y$ because x = pred(succ(x)) = pred(succ(y)) = y.

For example, we can rewrite the plus function in terms of the primitives of this algebra as

$$\text{plus}(x, y) = \text{if isZero}(x) \text{ then } y \text{ else succ}(\text{plus}(\text{pred}(x), y)).$$

We can also write the mult function in terms of the primitives of (10.11) together with the plus function as follows:

$$\text{mult}(x, y) = \text{if isZero}(x) \text{ then } 0 \text{ else plus}(\text{mult}(\text{pred}(x), y), y).$$

example **10.17 Less-Than**

Let's define the "less" relation on natural numbers using only the primitives of the algebra (10.11). To get an idea of how we might proceed, consider the following evaluation of the expression less(2, 4):

$$\text{less}(2, 4) = \text{less}(1, 3) = \text{less}(0, 2) = \text{true}.$$

We simply replace each argument by its predecessor until one of the arguments is zero. Therefore, less can be computed from a recursive definition such as the following:

$$\begin{aligned}
&\text{less}\,(0, 0) = \text{false},\\
&\text{less}\,(\text{succ}\,(x), 0) = \text{false},\\
&\text{less}\,(0, \text{succ}\,(y)) = \text{true},\\
&\text{less}\,(\text{succ}\,(x), \text{succ}\,(y)) = \text{less}\,(x, y)\,.
\end{aligned}$$

Using the if-then-else form, we obtain the following definition:

$$\begin{aligned}
\text{less}\,(x, y) = {} & \text{if isZero}\,(y) \text{ then false}\\
& \text{else if isZero}\,(x) \text{ then true}\\
& \text{else less}\,(\text{pred}\,(x), \text{pred}\,(y))\,.
\end{aligned}$$

end example

The following paragraphs describe several fundamental algebras of computer science. As we have said, they are also called abstract data types. The need for abstraction can be seen by considering questions like the following: What do lists and stacks have in common? How can we be sure that a queue is implemented correctly? How can we be sure that any data structure is implemented correctly? The answers to these questions depend on how we define the structures that we are talking about, without regard to any particular implementation.

10.3.2 Lists and Strings

Lists

Recall that the set of lists over a set A can be defined inductively by using the empty list, $\langle\ \rangle$, and the cons operation (with infix form ::) as constructors. If we denote the set of all lists over A by lists(A), we have the following inductive definition:

Basis: $\langle\ \rangle \in \text{lists}(A)$.

Induction: If $x \in A$ and $L \in \text{lists}(A)$, then $\text{cons}(x, L) \in \text{lists}(A)$.

The algebra of lists can be defined by the constructors $\langle\ \rangle$ and cons together with the primitive operations isEmptyL, head, and tail. With these operations we can describe the *list abstract data type* as the following algebra of lists over A:

Abstract Data Type of Lists

Carriers: lists(A), A, Boolean.

Operations: $\langle\ \rangle \in$ lists(A),
isEmptyL : lists(A)$\rightarrow$ Boolean,
cons : $A \times$ lists(A) $\rightarrow$ lists(A),
head : lists(A)$\rightarrow$ A,
tail : lists(A) $\rightarrow$ lists(A).

Axioms: isEmptyL($\langle\ \rangle$) = true,
isEmptyL(cons(x, L)) = false,
head(cons(x, L)) = x,
tail(cons(x, L)) = L.

Can all desired list functions be written in terms of the "primitive" operations of this algebra? The answer probably depends on the definition of "desired." For example, we saw in Chapter 3 that the following functions can be written in terms of the operations of the list algebra.

length:	lists(A) $\rightarrow \mathbb{N}$	Finds length of a list.
member:	$A\times$ lists(A) $\rightarrow$ Boolean	Tests membership in a list.
last:	lists(A) $\rightarrow A$	Finds last element of a list.
concatenate:	lists(A) $\times$ lists(A) $\rightarrow$ lists(A)	
putLast:	$A\times$ lists(A) $\rightarrow$ lists(A)	Puts element at right end.

Let's look at a couple of these functions to see whether we can implement them. Assume that all the operations in the signature of the list algebra are implemented. Then a definition for "length" can be written as follows:

$$\begin{aligned}\text{length}\,(L) = {} & \text{if isEmptyL}\,(L)\ \text{ then } 0\\ & \text{else } 1 + \text{length}(\text{tail}\,(L)\,).\end{aligned}$$

In this case the algebra $\langle\mathbb{N};\ +,\ 0\rangle$ must also be implemented for the length function to work properly.

Similarly, suppose we define "member" as follows:

$$\begin{aligned}\text{member}\,(a, L) = {} & \text{if isEmptyL}\,(L)\ \text{ then false}\\ & \text{else if } a = \text{head}\,(L)\ \text{ then true}\\ & \text{else member}\,(a, \text{tail}\,(L))\,.\end{aligned}$$

In this case the predicate "$a = \text{head}(L)$" must be computed. Thus an equality relation must be implemented for the carrier A.

As these two examples have shown, although we can define list functions in terms of the algebra of lists, we often need other algebras, such as $\langle \mathbb{N}; +, 0\rangle$, or other relations, such as equality on A.

Strings

Strings may look different than lists, but these structures have a lot in common. For example, they both have length, and their constructions are similar. For example, the set of all strings over an alphabet A can be defined inductively from the empty string, Λ, and the append operation to attach a letter to a string (which we'll denote by $\cdot$). Letting A^* denote the set of all strings over A, we have the following inductive definition:

Basis: $\Lambda \in A^*$.

Induction: If $x \in A$ and $s \in A^*$, then $x \cdot s \in A^*$.

The algebra of strings can be defined by the constructors Λ and append together with the primitive operations isEmptyS, headS, and tailS. With these operations we can describe the *string abstract data type* as the following algebra of strings over A:

Abstract Data Type of Strings

Carriers: A, A^*, Boolean.

Operations: $\Lambda \in A^*$,

isEmptyS : $A^* \to$ Boolean,

$\cdot : A \times A^* \to A^*$,

headS : $A^* \to A$,

tailS : $A^* \to A^*$.

Axioms: isEmptyS(Λ) = true,

isEmptyS$(a \cdot s)$ = false,

headS$(a \cdot s) = a$,

tailS$(a \cdot s) = s$.

When working with strings, we want to be able to combine strings, compare strings, and so on. We can define functions to accomplish these things using the string algebra. For example, let's write a definition for the "cat" function to combine two strings. For example, cat(cb, aba) = $cbaba$. Cat has type $A^* \times A^* \to A^*$ and can be defined as follows:

$$\begin{aligned}\text{cat}(s,t) = {} & \text{if isEmptyS}(s) \text{ then } t \\ & \text{else headS}(s) \cdot \text{cat}(\text{tailS}(s), t).\end{aligned}$$

10.3.3 Stacks and Queues

Stacks

A ***stack*** is a structure satisfying the LIFO property of last in, first out. In other words, the last element input is the first element output. The main stack operations are *push*, which pushes a new element onto a stack; *pop*, which removes the top element from a stack; and *top*, which examines the top element of a stack. We also need an indication of when a stack is empty.

Let's describe the ***stack abstract data type*** as an algebra. For any set A, let Stks[A] denote the set of stacks whose elements are from A. We'll include error messages in our description for those cases in which the operators are not defined. Here's the algebra.

Abstract Data Type of Stacks

Carriers:	A, Stks[A], Boolean, Errors.
Operations:	emptyStk $\in$ Stks[A],
	isEmptyStk : Stks[A] $\to$ Boolean,
	push : $A \times$ Stks[A] $\to$ Stks[A],
	pop : Stks[A] $\to$ Stks[A] $\cup$ Errors,
	top : Stks[A] $\to$ $A \cup$ Errors.
Axioms:	isEmptyStk(emptyStk) = true,
	isEmptyStk(push(a, s)) = false,
	pop(push(a, s)) = s,
	pop(emptyStk) = stackError,
	top(push(a, s)) = a,
	top(emptyStk) = valueError.

Notice the similarity between the stack algebra and the list algebra. In fact, we can implement the stack algebra as a list algebra by assigning the following meanings to the stack symbols:

$$\begin{aligned}\text{Stks}\,[A] &= \text{lists}\,(A)\,,\\ \text{emptyStk} &= \langle\ \rangle\,,\\ \text{isEmptyStk} &= \text{isEmptyL},\\ \text{push} &= \text{cons},\\ \text{pop} &= \text{tail},\\ \text{top} &= \text{head}.\end{aligned}$$

To prove that this implementation is correct, we need to show that the axioms of a stack are true for the above assignment. They are all trivial. For example, the proof of the third axiom is a one-liner:

$$\text{pop}(\text{push}(a,\ s)) = \text{tail}(\text{cons}(a,\ s)) = s.\ \text{QED}$$

example 10.18 Evaluating a Postfix Expression

Let's look at the general approach to evaluate an arithmetic expression represented in postfix notation. For example, the postfix expression $abc+-$ can be evaluated by pushing a, b, and c onto a stack. Then b and c are popped, and the value $b + c$ is pushed onto the stack. Finally, a and $b + c$ are popped, and the value $a - (b + c)$ is pushed. We'll assume that all operators are binary and that there is a function "val," which takes an operator and two operands and returns the value of the operator applied to the two operands.

The general algorithm for evaluating a postfix expression can be given as follows, where the initial call has the form post(L, $\langle\ \rangle$) and L is the list representation of the postfix expression:

$$\begin{aligned}\text{post}\,(\langle\ \rangle\,, \text{stk}) &= \text{top}(\text{stk})\\ \text{post}\,(x :: t, \text{stk}) &= \text{if } x \text{ is an argument then}\\ &\qquad\quad \text{post}\,(t, \text{push}\,(x, \text{stk}))\\ &\quad\ \text{else}\ \ \{x \text{ is an operator}\}\\ &\qquad\quad \text{post}\,(t, \text{eval}\,(x, \text{stk}))\,,\end{aligned}$$

where eval is defined by the equation

$$\text{eval}(\text{op},\ \text{push}(a,\ \text{push}(b,\ \text{stk}))) = \text{push}(\text{val}(b,\ \text{op},\ a),\ \text{stk}).$$

For example, we'll evaluate the expression post($\langle 2, 5, +\rangle$, $\langle\ \rangle$).

$$\begin{aligned}\text{post}\left(\langle 2,5,+\rangle,\langle\ \rangle\right) &= \text{post}\left(\langle 5,+\rangle,\langle 2\rangle\right)\\ &= \text{post}\left(\langle +\rangle,\langle 5,2\rangle\right)\\ &= \text{post}\left(\langle\ \rangle,\text{eval}\left(+,\langle 5,2\rangle\right)\right)\\ &= \text{top}\left(\text{eval}\left(+,\langle 5,2\rangle\right)\right)\\ &= \text{top}\left(\text{push}\left(\text{val}\left(2,+,5\right),\langle\ \rangle\right)\right)\\ &= \text{val}\left(2,+,5\right)\\ &= 7.\end{aligned}$$

end example

Queues

A *queue* is a structure satisfying the FIFO property of first in, first out. In other words, the first element input is the first element output. So a queue is a fair waiting line. The main operations on a queue involve adding a new element, examining the front element, and deleting the front element.

To describe the *queue abstract data type* as an algebra, we'll let A be a set and Q[A] be the set of queues over A. Here's the algebra.

Abstract Data Type of Queues

Carriers: A, $Q[A]$, Boolean.

Operations: emptyQ $\in Q[A]$,

isEmptyQ : $Q[A] \to$ Boolean,

addQ : $A \times Q[A] \to Q[A]$,

frontQ : $Q[A] \to A$,

delQ : $Q[A] \to Q[A]$.

Axioms: isEmptyQ(emptyQ) = true,

isEmptyQ(addQ(a, q)) = false,

frontQ(addQ(a, q)) = if isEmptyQ(q) then a
else frontQ(q),

delQ(addQ(a, q)) = if isEmptyQ(q) then q
else addQ(a, delQ(q)).

Although we haven't stated it in the axioms, an error will occur if either frontQ or delQ is applied to an empty queue.

Suppose we represent a queue as a list. For example, the list $\langle a, b\rangle$ represents a queue with a at the front and b at the rear. If we add a new item c to this queue, we obtain the queue $\langle a, b, c\rangle$. So addQ(c, $\langle a, b\rangle$) = $\langle a, b, c\rangle$. Thus addQ can be implemented as the putLast function. The implementation of a queue algebra as a list algebra can be given as follows:

$$\begin{aligned} Q[A] &= \text{lists}(A)\\ \text{emptyQ} &= \langle\ \rangle,\\ \text{isEmptyQ} &= \text{isEmptyL},\\ \text{frontQ} &= \text{head},\\ \text{delQ} &= \text{tail},\\ \text{addQ} &= \text{putLast}. \end{aligned}$$

The proof of correctness of this implementation is more interesting (not trivial) because two queue axioms include conditionals, and putLast is written in terms of the list primitives. For example, we'll prove the correctness of the third axiom for the algebra of queues, leaving the proof of the fourth axiom as an exercise. Since the third axiom is an if-then-else statement, we'll consider two cases:

Case 1: Assume that $q = \text{emptyQ}$. In this case the axiom becomes

$$\begin{aligned} \text{frontQ}(\text{addQ}(a, \text{emptyQ})) &= \text{head}(\text{putLast}(a, \text{emptyQ}))\\ &= \text{head}(a :: \text{emptyQ})\\ &= a. \end{aligned}$$

Case 2: Assume that $q \neq \text{emptyQ}$. In this case the axiom becomes

$$\begin{aligned} \text{frontQ}(\text{addQ}(a, q)) &= \text{head}(\text{putLast}(a, q))\\ &= \text{head}(\text{head}(q) :: \text{putLast}(a, \text{tail}(q)))\\ &= \text{head}(q)\\ &= \text{frontQ}(q). \end{aligned}$$

example 10.19 The Append Function

Let's use the queue algebra to define the append function, apQ, which joins two queues together. It can be written in terms of the primitive operations of a queue algebra as follows:

$$\begin{aligned} \text{apQ}(x, y) = {} & \text{if isEmptyQ}(y) \text{ then } x\\ & \text{else apQ}(\text{addQ}(\text{frontQ}(y), x), \text{delQ}(y)). \end{aligned}$$

For example, suppose $x = \langle a, b\rangle$ and $y = \langle c, d\rangle$ are two queues, where a is the front of x and c is the front of y. We'll evaluate the expression apQ(x, y).

$$\begin{aligned}\text{apQ}(x,y) &= \text{apQ}(\langle a,b\rangle, \langle c,d\rangle)\\ &= \text{apQ}(\langle a,b,c\rangle, \langle d\rangle)\\ &= \text{apQ}(\langle a,b,c,d\rangle, \langle\ \rangle)\\ &= \langle a,b,c,d\rangle .\end{aligned}$$

end example

example **10.20 Decimal to Binary**

Let's convert a natural number to a binary number and represent the output as a queue of binary digits. Let bin(n) represent the queue of binary digits representing n. For example, we should have bin(4) = $\langle 1, 0, 0\rangle$, assuming that the front of the queue is the head of the list. Let's get to the definition.

If $n = 0$ or $n = 1$, we should return the queue $\langle n\rangle$, which is constructed by addQ(n, emptyQ). If n is not 0 or 1, then we should return the queue addQ(n mod 2, bin(floor(n/2))). In other words, we can define bin as follows:

$$\begin{aligned}\text{bin}(n) = {}&\text{if } n = 0 \text{ or } n = 1 \text{ then}\\ &\qquad \text{addQ}(n, \text{emptyQ})\\ &\text{else}\\ &\qquad \text{addQ}(n \bmod 2, \text{bin}(\text{floor}(n/2))).\end{aligned}$$

We leave it as an exercise to check that bin works. For example, try to evaluate the expression bin(4) to see whether you get the list $\langle 1, 0, 0\rangle$.

end example

10.3.4 Binary Trees and Priority Queues

Binary Trees

Let $B[A]$ denote the set of binary trees over a set A. The main operations on binary trees involve constructing a tree, picking the root, and picking the left and right subtrees. If $a \in A$ and l, $r \in B[A]$, let tree(l, a, r) denote the tree whose root is a, whose left subtree is l, and whose right subtree is r. We can describe the *binary tree abstract data type* as the following algebra of binary trees:

Abstract Data Type of Binary Trees

Carriers: A, $B[A]$, Boolean.

Operations: emptyTree $\in B[A]$,

isEmptyTree : $B[A] \to$ Boolean,

root : $B[A] \to A$,

tree : $B[A] \times A \times B[A] \to B[A]$,

left : $B[A] \to B[A]$,

right : $B[A] \to B[A]$.

Axioms: isEmptyTree(emptyTree) = true,

isEmptyTree(tree(l, a, r)) = false,

left(tree(l, a, r)) = l,

right(tree(l, a, r)) = r,

root(tree(l, a, r)) = a.

Although we haven't stated it in the axioms, an error will occur if the functions left, right, and root are applied to the empty tree. Next, we'll give a few examples to show how useful functions can be constructed from the basic tree operations.

example

10.21 Nodes and Depth

We'll look at two typical functions, "count" and "depth." Count returns the number of nodes in a binary tree. Its type is $B[A] \to \mathbb{N}$, and its definition follows:

$$\begin{aligned} \text{count}(t) = {} & \text{if isEmptyTree}(t) \text{ then } 0 \\ & \text{else } 1 + \text{count}(\text{left}(t)) + \text{count}(\text{right}(t)). \end{aligned}$$

Depth returns the length of the longest path from the root to the leaves of a binary tree. Assume that an empty binary tree has depth -1. Its type is $B[A] \to \mathbb{Z}$, and its definition follows:

$$\begin{aligned} \text{depth}(t) = {} & \text{if isEmptyTree}(t) \text{ then } -1 \\ & \text{else } 1 + \max(\text{depth}(\text{left}(t)), \text{depth}(\text{right}(t))). \end{aligned}$$

end example

example

10.22 Inorder Traversal

Suppose we want to write a function "inorder" to perform an inorder traversal of a binary tree and place the nodes in a queue. So we want to define a function of type $B[A] \to Q[A]$. For example, we might use the following definition:

$$\begin{aligned}\text{inorder}(t) = {} & \text{if isEmptyTree}(t) \text{ then emptyQ} \\ & \text{else apQ}(\text{addQ}(\text{root}(t), \text{inorder}(\text{left}(t))), \text{inorder}(\text{right}(t))).\end{aligned}$$

We'll leave the preorder and postorder traversals as exercises.

end example

Priority Queues

A *priority queue* is a structure satisfying the BIFO property: best in, first out. For example, a stack is a priority queue if we let Best = Last. Similarly, a queue is a priority queue if we let Best = First. The main operations of a priority queue involve adding a new element, accessing the best element, and deleting the best element.

Let $P[A]$ denote the set of priority queues over A. If $a \in A$ and $p \in P[A]$, then insert(a, p) denotes the priority queue obtained by adding a to p. We can describe the *priority queue abstract data type* as the following algebra:

Carriers: A, $P[A]$, Boolean.

Operations: emptyP $\in P[A]$,
isEmptyP : $P[A] \to$ Boolean,
better : $A \times A \to$ Boolean,
best : $P[A] \to A$,
insert : $A \times P[A] \to P[A]$,
delBest : $P[A] \to P[A]$.

We'll note here that we are assuming that the function "better" is a binary relation on A. Now for the axioms:

Axioms: isEmptyP(emptyP) = true,
isEmptyP(insert(a, p)) = false,

$$\begin{aligned}\text{best}(\text{insert}(a, p)) = {} & \text{if isEmptyP}(p) \text{ then } a \\ & \text{else if better}(a, \text{best}(p)) \text{ then } a \\ & \text{else best}(p),\end{aligned}$$

$$\begin{aligned}\text{delBest}(\text{insert}(a, p)) = {} & \text{if isEmptyP}(p) \text{ then emptyP} \\ & \text{else if better}(a, \text{best}(p)) \text{ then } p \\ & \text{else insert}(a, \text{delBest}(p)).\end{aligned}$$

We should note that the operations best and delBest are defined only on nonempty priority queues. Priority queues can be implemented in many different ways, depending on the definitions of "better" and "best" for the set A.

To show the power of priority queues, we'll write a sorting function that sorts the elements of a priority queue into a sorted list. The initial call to sort the priority queue p is sort(p, $\langle\ \rangle$). The definition can be written as follows:

$$\begin{aligned}\text{sort}(p, L) = {} & \text{if isEmptyP}(p) \text{ then } L \\ & \text{else sort}(\text{delBest}(p), \text{best}(p) :: L).\end{aligned}$$

Exercises

Natural Numbers

1. The *monus* operation on natural numbers is like subtraction, except that it always gives a natural number as a result. An informal definition of monus can be written as follows:

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

Write down a recursive definition of monus that uses only the primitive operations isZero, succ, and pred.

2. The exponentiation function is defined by exp(a, b) $= a^b$. Write down a recursive definition of exp that uses primitive operations or functions that are defined in terms of the primitive operations on the natural numbers. *Note:* Assume that exp(0, 0) = 0.

Lists

3. Use the algebra of lists to write a definition of the function "reverse" to reverse the elements of a list. For example, reverse($\langle x, y, z\rangle$) = $\langle z, y, x\rangle$.

4. Use lists to describe an implementation of the algebra of strings over some alphabet.

5. Write an algebraic specification for general lists over a set A (where the elements of a list may also be lists).

6. Use the algebra of lists to write a definition for the "flatten" function that takes a general list over a set A and returns the list of its elements from A. For example, flatten($\langle\langle a, b\rangle, c, d\rangle$) = $\langle a, b, c, d\rangle$. *Hint:* Assume that there is a function isAtom to check whether its argument is an atom (not a list). Also assume that the other list operations work on general lists.

7. Evaluate the expression post($\langle 4, 5, -, 2, +\rangle$, $\langle\ \rangle$) by unfolding the definition in Example 10.18.

8. Evaluate the expression bin(4) by unfolding the definition in Example 10.20.

Binary Trees

9. Write down a definition for the function "preorder," which performs a preorder traversal of a binary tree and places the node values in a queue.

10. Write down a definition for the function "postorder," which performs a postorder traversal of a binary tree and places the node values in a queue.

Stacks and Queues

11. Find a descriptive name for the "mystery" function f, which has the type $A \times \text{Stks}[A] \to \text{Stks}[A]$ and is defined by the following equations:

$$\begin{aligned} f\,(a, \text{emptyStk}) &= \text{emptyStk}, \\ f\,(a, \text{push}\,(a, s)) &= f\,(a, s)\,, \\ f\,(a, \text{push}\,(b, s)) &= \text{push}\,(b, f\,(a, s)) \quad \text{if } a \neq b. \end{aligned}$$

12. Find a descriptive name for the "mystery" function f, which has type $Q[A] \to Q[A]$ and is defined as follows:

$$\begin{aligned} f\,(q) = {} & \text{if isEmptyQ}\,(q) \;\text{ then } q \\ & \text{else addQ}\,(\text{frontQ}\,(q)\,, f\,(\text{delQ}\,(q)))\,. \end{aligned}$$

13. A *deque*, pronounced "deck," is a double-ended queue in which insertions and deletions can be made at either end of the deque. Write down an algebraic specification for deques over a set A.

14. For the list implementation of a queue, prove the correctness of the following axiom:

$$\begin{aligned} \text{delQ}\,(\text{addQ}\,(a, q)) = {} & \text{if isEmptyQ}\,(q) \;\text{ then } q \\ & \text{else addQ}\,(a, \text{delQ}\,(q))\,. \end{aligned}$$

15. Implement a queue by using the operations of a deque. Prove the correctness of your implementation.

16. Suppose the "better" function used in a priority queue has the following type definition:

$$\text{better} : A \times A \to A.$$

How would the axioms change? Do we need any new operations?

Proofs and Challenges

17. Consider the following two definitions for adding natural numbers, where p and s denote the predecessor and successor operations.

$$\begin{aligned}\text{plus}(x,y) &= \text{if } x=0 \text{ then } y \text{ else } s(\text{plus}(p(x),y)),\\ \text{add}(x,y) &= \text{if } x=0 \text{ then } y \text{ else } \text{add}(p(x),s(y)).\end{aligned}$$

   a. Use induction to prove that plus(x, $s(y)$) = s(plus(x, y)) for all x, $y \in \mathbb{N}$.
   b. Use induction to prove that plus(x, y) = add(x, y) for all x, $y \in \mathbb{N}$. *Hint:* Part (a) can be useful.

18. Use induction to prove the following property over a queue algebra, where apQ is the append function defined in Example 10.19.

$$\text{apQ}(x, \text{addQ}(a, y)) = \text{addQ}(a, \text{apQ}(x, y)).$$

Hint: To simplify notation, let $x{:}a$ denote addQ(a, x). Then the equation becomes apQ(x, $y{:}a$) = apQ(x, y)$:a$.

19. Use induction to prove the following property over a queue algebra, where apQ is the append function defined in Example 10.19.

$$\text{apQ}(x, \text{apQ}(y, z)) = \text{apQ}(\text{apQ}(x, y), z).$$

Hint: Exercise 18 may be helpful.

10.4 Computational Algebras

In this section we present some important examples of algebras that are useful in the computation process. First we'll look at relational algebra as a tool for representing relational databases. Then we'll discuss functional algebra as a tool not only for programming, but for reasoning about programs.

10.4.1 Relational Algebras

Relations can be combined in various ways to build new relations that solve problems. An algebra is called a *relational algebra* if its carrier is a set of relations. We'll discuss three useful operations on relations: *select*, *project*, and *join.* Each of these operations builds a new relation by selecting certain tuples, by eliminating certain attributes, or by combining attributes of two relations. We'll motivate the definitions with some examples.

Rooms

Place	Seats	Boardtype	Computer
CH171	80	Chalk	No
HH101	250	No	Yes
SC211	35	White	Yes
CH301	90	Chalk	Yes
...	...	...	...

Figure 10.11 Classooms in a school.

Let *Rooms* be the relation with attributes {Place, Seats, Boardtype, Computer} to describe classrooms in a college. For example, Figure 10.11 shows a few sample entries for Rooms.

Notice that Rooms can be represented as a relation consisting of a set of 4-tuples as follows:

$$\begin{aligned}\text{Rooms} = \{ & (\text{CH171}, 80, \text{Chalk}, \text{No}),\\ & (\text{HH101}, 250, \text{No}, \text{Yes}),\\ & (\text{SC211}, 35, \text{White}, \text{Yes}),\\ & (\text{CH301}, 90, \text{Chalk}, \text{Yes}), \ldots\}.\end{aligned}$$

The Select Operation

The *select* operation on a relation forms a new relation that is a subset of the relation consisting of those tuples that have a common value in one of the attributes.

For example, suppose that we want to construct the relation A of tuples that represent all the rooms with chalk boards. In other words, we want to select from Rooms those tuples that have Boardtype equal to Chalk. We'll represent this by the notation

$$A = \text{select}(\text{Rooms}, \text{Boardtype}, \text{Chalk}).$$

The value of this expression is

$$A = \{(\text{CH171}, 80, \text{Chalk}, \text{No}), (\text{CH301}, 90, \text{Chalk}, \text{Yes}), \ldots\}.$$

example **10.23 Selecting Tuples**

Suppose we want to construct the relation B of tuples that represent the rooms with chalk boards and computers. In this case we can select the tuples from the relation A.

$$\begin{aligned}B &= \text{select}(A, \text{Computer}, \text{Yes})\\ &= \text{select}(\text{select}(\text{Rooms}, \text{Boardtype}, \text{Chalk}), \text{Computer}, \text{Yes})\\ &= \{(\text{CH301}, 90, \text{Chalk}, \text{Yes}), \ldots\}.\end{aligned}$$

end example

The Project Operation

The *project* operation on a relation forms a new relation consisting of tuples indexed by a subset of the attributes of the relation.

For example, suppose that we want to construct the relation *Size* of tuples with only the two attributes Place and Seats. In other words, we want to restrict ourselves to the first and second columns of the table for Rooms. We'll represent this by the notation

$$\text{Size} = \text{project(Rooms, \{Place, Seats\})}.$$

The value of this expression is

$$\text{Size} = \{(\text{CH171}, 80), (\text{HH101}, 250), (\text{SC211}, 35), (\text{CH301}, 90), \ldots\}.$$

example **10.24 Specific Properties**

Here are a few more questions that ask for specific properties about the Rooms relation.

1. What rooms have chalk boards?

   project(select(Rooms, Boardtype, Chalk), {Place})
   = {(CH171), (CH301), ...}.

2. How large are the rooms with computers?

   project(select(Rooms, Computer, Yes), {Place, Seats})
   = {(HH101, 250), (SC211, 35), (CH301, 90), ...}.

3. What kind of board is in SC211?

   project(select(Rooms, Place, SC211), {Boardtype})
   = {(White)}.

end example

The Join Operation

The *join* operation on two relations forms a new relation consisting of tuples that are indexed by the union of the attributes of the two relations. The new tuples in the join are constructed from pairs of tuples whose values agree on the common attributes of the two relations.

For example, let *Channel* and *Program* be two relations with attributes {Station, Satellite, Cable} and {Station, Type}, respectively, that describe information about television networks. Figure 10.12 shows a few sample entries for the two relations.

Suppose we want to join the two relations into a single relation called TV with attributes Station, Satellite, Cable, and Type. We'll represent this operation by the notation

$$\text{TV} = \text{join(Channel, Program)}.$$

Channel

Station	Satellite	Cable
AMC	130	48
CNN	200	96
TCM	132	54
ESPN	140	32

Program

Station	Type
AMC	Movie
CNN	News
TCM	Movie
ESPN	Sports

Figure 10.12 TV channels and programs.

example

10.25 TV Questions

Now we can answer some TV questions. For example, what are the cable movie channels? One solution is to select the tuples in TV that have Movie as the Type attribute. Then project onto the Cable attribute:

$$\text{project}(\text{select}(\text{TV, Type, Movie}), \{\text{Cable}\})$$

The expression evaluates to the channel numbers $\{(48), (54), \ldots\}$.

For another example, what type of programming is on satellite channel 140? One solution is to select the tuples in TV that have 140 as the Satellite attribute. Then project onto the Type attribute:

$$\text{project}(\text{select}(\text{TV, Satellite, 140}), \{\text{Type}\}).$$

The expression evaluates to $\{(\text{Sports})\}$.

end example

example

10.26 Class Schedules

Let *Schedule* be the class schedule with attributes {Dept, Course, Section, Credit, Time, Day, Place, Teacher}. Figure 10.13 shows a few sample entries.

Each entry of the table can be represented as a tuple. For example, the first row of Schedule can be represented as the tuple

$$(\text{CS, 252, 1, 4, 1600–1750, TTh, CH171, Hein}).$$

Schedule

Dept	Course	Section	Credit	Time	Day	Place	Teacher
CS	252	1	4	1600-1750	TTh	CH171	Hein
CS	252	2	4	1200-1350	MW	SC211	Jones
Mth	201	1	4	1000-1150	TTh	CH301	Appleby
Mth	256	1	4	1400-1550	MW	NH356	Ames
EE	300	1	4	0800-0950	TTh	SC211	Brand

Figure 10.13 A class schedule.

Here are some sample questions and answers.

1. What is the mathematics schedule?

   select(Schedule, Dept, Mth).

2. What mathematics classes meet TTh?

   select(select(Schedule, Dept, Mth), Day, TTh)

   = {(Mth, 201, 1, 4, 1000–1150, TTh, NH325, Appleby), . . . }.

3. What are the times and days that CS 252 is taught? If we want the set

   {(1600–1750, TTh), (1200–1350, MW)},

   then we can use the following expression:

   project(select(select(Schedule, Dept, CS), Course, 252), {Time, Day})

   = {(1600–1750, TTh), (1200-1350, MW)}.

4. What classes are in rooms with computers?

   project(join(Rooms, Schedule), {Dept, Course, Section})

   = {(CS, 252, 2), (Mth, 201, 1), (EE, 300, 1), . . . }.

end example

Formal Definitions of Select, Project, and Join

Let R be a relation, A an attribute of R, and a a possible value of A. The relation consisting of all tuples in R with attribute A having value a is denoted by select(R, A, a) and is defined as follows:

Select Operation

$$\text{select}(R, A, a) = \{t \mid t \in R \text{ and } t(A) = a\}.$$

For example, using the Rooms relation in Figure 10.11 we have

select(Rooms, Boardtype, Chalk)

= {t | $t \in$ Rooms and t(Boardtype) = Chalk}.

= {(CH171, 80, Chalk, No), (CH301, 90, Chalk, Yes), . . . }.

If A and a are fixed, then select(R, A, a) is sometimes denoted by $\text{select}_{A=a}(R)$.

If X is a subset of the set of attributes of the relation R, then the project operation of R on X is denoted project(R, X) and consists of all tuples indexed by X constructed from the tuples of R. In formal terms we have the following definition:

Project Operation

$$\text{project}(R, X) = \{s \mid \text{there exists } t \in R \text{ such that } s(A) = t(A) \text{ for all } A \in X\}.$$

For example, using the Rooms relation in Figure 10.11 we have

$$\begin{aligned}&\text{project}(\text{Rooms}, \{\text{Place}, \text{Seats}\})\\ &\quad = \{s \mid \text{there exists } t \in \text{Rooms such that}\\ &\qquad\quad s(\text{Place}) = t(\text{Place}) \text{ and } s(\text{Seats}) = t(\text{Seats})\}\\ &\quad = \{(\text{CH171}, 80), (\text{HH101}, 250), (\text{SC211}, 35), (\text{CH301}, 90), \ldots\}.\end{aligned}$$

If X is fixed, then project(R, X) is sometimes denoted by project$_X(R)$.

Let R and S be two relations with attribute sets I and J, respectively. The *join* of R and S is the set of all tuples over the attribute set $I \cup J$ that are constructed from R and S by "joining" those tuples with equal values on the common attribute set $I \cap J$. We denote the join of R and S by join(R, S). Here's the formal definition:

Join Operation

$$\begin{aligned}\text{join}(R, S) = \{t \mid &\text{there exist } r \in R \text{ and } s \in S \text{ such that}\\ &t(A) = r(A) \text{ for all } A \in I \text{ and}\\ &t(B) = s(B) \text{ for all } B \in J\}.\end{aligned}$$

There are two special cases. Let R and S be two relations with attribute sets I and J. If $I \cap J = \varnothing$, then join(R, S) is obtained by concatenating all pairs of tuples in $R \times S$. For example, if we have tuples $(a, b) \in R$ and $(c, d, e) \in S$, then $(a, b, c, d, e) \in$ join(R, S). If $I = J$, then join(R, S) $= R \cap S$.

A Relational Algebra

Now we have the ingredients of a relational algebra. The carrier is the set of all possible relations, and the three operations are select, project, and join. We should remark that join(R, S) is often denoted by $R \bowtie S$. The properties of this relational algebra are too numerous to mention here. But we'll list a few properties that can be readily verified from the definitions.

$$\begin{aligned}&\text{select}_{A=a}\,(\text{select}_{B=b}\,(R)) = \text{select}_{B=b}\,(\text{select}_{A=a}\,(R))\,,\\ &R \bowtie R = R,\\ &(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)\,,\\ &\text{project}_X\,(\text{select}_{A=a}\,(R)) = \text{select}_{A=a}\,(\text{project}_X\,(R)) \quad (\text{where } A \in X)\end{aligned}$$

If R and S have the same set of attributes, then the select operation has some nice properties when combined with the set operations: $\cup$, $\cap$, and $-$. For example, we have the following properties:

$$\begin{aligned}\text{select}_{A=a}\,(R \cup S) &= \text{select}_{A=a}\,(R) \cup \text{select}_{A=a}\,(S)\,,\\ \text{select}_{A=a}\,(R \cap S) &= \text{select}_{A=a}\,(R) \cap \text{select}_{A=a}\,(S)\,,\\ \text{select}_{A=a}\,(R - S) &= \text{select}_{A=a}\,(R) - \text{select}_{A=a}\,(S)\,.\end{aligned}$$

There are many other useful operations on relations, some of which can be defined in terms of the ones we have discussed. Relational algebra provides a set of tools for constructing, maintaining, and accessing databases.

10.4.2 Functional Algebras

From a programming point of view, a *functional algebra* consists of functions together with operations to combine functions in order to process data objects. Let's look at a particular functional algebra that is both a programming language and an algebra for reasoning about programs.

FP: A Functional Algebra

The correctness problem for programs can sometimes be solved by showing that the program under consideration is equivalent to another program that we "know" is correct. Methods for showing equivalence depend very much on the programming language. The FP language was introduced by Backus [1978]. FP stands for *functional programming*, and it is a fundamental example of a programming language that allows us to reason about programs in the programming language itself. To do this, we need a set of rules that allow us to do some reasoning. In this case the rules are axioms in the algebra of FP programs.

FP functions are defined on a set of objects that include atoms (numbers, and strings of characters), and lists. To apply an FP function f to an object x, we write down $f{:}x$ instead of the familiar $f(x)$. To compose two FP functions f and g, we write f @ g instead of the familiar $f \circ g$. Suppose we have the following definition for an if-then-else function:

$$f(x) = \text{if } a(x) \text{ then } b(x) \text{ else } c(x).$$

We would make f into an FP function by writing

$$f = a \rightarrow b;\ c.$$

We evaluate $f{:}x$ just like the evaluation of an if-then-else statement:

$$f{:}x = (a \rightarrow b;\ c){:}x = a{:}x \rightarrow b{:}x;\ c{:}x.$$

A function can be defined as a tuple of functions. For example, the following expression defines f as a 3-tuple of functions:

$$f = [g,\ h,\ k].$$

In this case $f{:}x = \langle g{:}x,\ \ h{:}x,\ \ k{:}x\rangle$. A constant function has the form $f = \sim c$, where $f{:}x = c$ for all objects x.

Our objective is to get the flavor of the language to see its algebraic nature. So we'll describe some operations and axioms of an algebra of FP programs. We'll limit the operations to those that will be useful in the examples and exercises. We'll also include a few axioms to show some useful relationships between composition, tupling, and if-then-else.

Operations to construct new functions:

@	composition (e.g., f @ g),
$\rightarrow$	if-then-else (e.g., $p \rightarrow a; b$),
$[\ldots]$	tuple of functions (e.g., $[f, g, h]$),
$\sim$	constant (e.g., ~ 2).

Primitive operations:

id	the identity function,
hd, tl	head and tail,
apndl, apndr	cons and consR,
1, 2, ...	selectors (e.g., $2{:}\langle a, b, c\rangle = b$),
and, or, not	Boolean operations,
null	test for empty list,
atom	test for an atom,
$\langle\ \rangle$	empty list,
?	undefined symbol,
eq	test for equality of two atoms,
$<$, $>$, $+$, $-$, $*$, $/$	arithmetic relations and operations.

Axioms:

f @ $(a \rightarrow b\ ;\ c) = a \rightarrow f$ @ $b\ ;\ f$ @ c,

$(a \rightarrow b\ ;\ c)$ @ $d = a$ @ $d \rightarrow b$ @ $d\ ;\ c$ @ d,

$[f_1, \ldots, f_n]$ @ $g = [f_1$ @ $g, \ldots, f_n$ @ $g]$,

f @ $[\ldots, (a \rightarrow b\ ;\ c), \ldots] = a \rightarrow f$ @ $[\ldots, b, \ldots]\ ;\ f$ @ $[\ldots, c, \ldots]$.

Now we'll give some examples of FP program definitions. In the last example we'll use FP algebra to prove the equivalence of two FP programs.

example

10.27 Testing for Zero

Let eq0 be the function that tests its argument for zero. An FP definition for eq0 can be written as follows, where eq tests equality of atoms.

$$\text{eq0} = \text{eq} \ @ \ [\text{id}, \sim 0].$$

For example, we'll evaluate the expression eq0 : 3 as follows:

$$\text{eq0} : 3 = \text{eq} \ @ \ [\text{id}, \sim 0] : 3 = \text{eq} : \langle \text{id} : 3, \sim 0 : 3 \rangle = \text{eq} : \langle 3, 0 \rangle = \text{false}.$$

end example

example

10.28 Subtracting 1

Let sub1 be the function that subtracts 1 from its argument. An FP definition for sub1 can be written as follows, where − is subtraction.

$$\text{sub1} = - \ @ \ [\text{id}, \sim 1].$$

For example, we'll evaluate the expression sub1 : 4 as follows:

$$\text{sub1} : 4 = - \ @ \ [\text{id}, \sim 1] : 4 = - : \langle \text{id} : 4, \sim 1 : 4 \rangle = - : \langle 4, 1 \rangle = 3.$$

end example

example

10.29 Length of a List

Let "length" be the function that calculates the length of a list. An informal if-then-else definition of length can be written as follows.

$$\text{length}(x) = \text{if } x = \langle\ \rangle \text{ then } 0 \text{ else } 1 + \text{length}(\text{tail}(x)).$$

The corresponding FP definition can be written as follows, where null tests for the empty list and tl computes the tail of a list:

$$\text{length} = \text{null} \rightarrow \sim 0;\ + \ @ \ [\sim 1, \text{length} \ @ \ \text{tl}].$$

For example, we'll evaluate the expression length : $\langle a \rangle$.

$$\begin{aligned}
\text{length} : \langle a \rangle &= (\text{null} \to \sim 0; + @[\sim 1, \text{length} @ \text{tl}]) : \langle a \rangle \\
&= \text{null} : \langle a \rangle \to \sim 0 : \langle a \rangle; + @ [\sim 1, \text{length}@\text{tl}] : \langle a \rangle \\
&= \text{false} \to 0; + : \langle 1, \text{ length}: \langle\, \rangle \rangle \\
&= + : \langle 1, \text{ length}: \langle\, \rangle \rangle \\
&= + : \langle 1, (\text{null} \to \sim 0; +@ [\sim 1, \text{ length} @ \text{tl}]) : \langle\, \rangle \rangle \\
&= + : \langle 1, \text{ true} \to 0; +@ [\sim 1, \text{ length} @ \text{tl}] : \langle\, \rangle \rangle \\
&= + : \langle 1, 0 \rangle \\
&= 1.
\end{aligned}$$

end example

example

10.30 An Equivalence Proof

An FP program to compute $n!$ can be constructed directly from the following recursive definition.

$$\text{fact}(x) = \text{if } x = 0 \text{ then } 1 \text{ else } x * \text{fact}(x - 1).$$

The FP version of fact is

$$\text{fact} = \text{eq0} \to \sim 1; * @ [\text{id}, \text{fact} @ \text{sub1}].$$

An alternative FP program to compute $n!$ can be defined as follows.

$$\text{newfact} = g @ [\sim 1, \text{id}],$$

where g is the FP program defined by

$$g = \text{eq0} @ 2 \to 1; g @ [*, \text{sub1} @ 2].$$

Notice that g is iterative because it has a tail-recursive form (i.e., it has the form $g = a \to b;\ g @ d$), which can be replaced by a loop. Therefore, newfact is also iterative. So newfact may be more efficient than fact. To prove the two programs are equivalent, we'll need the following relation involving g:

$$* @ [a,\ g @ [b,\ c]] = g @ [* @ [a,\ b],\ c], \tag{10.12}$$

where a, b , and c are functions that return natural numbers. We'll leave the proof of (10.12) as an exercise. Now we can prove that newfact = fact.

Proof: If the input to either function is 0, then eq0 is true, which gives us the base case fact:0 = newfact:0. Now we'll make the induction assumption that fact @ sub1 = newfact @ sub1 and show that newfact = fact. Starting with newfact, we have the following sequence of algebraic equations:

newfact	$= g$ @ [~1, id]	(definition)
	$=$ (eq0 @ 2 → 1; g @ [∗, sub1 @ 2]) @ [~1, id]	(definition)
	$=$ eq0 @ id → ~1; g @ [∗ @ [~1, id], sub1 @ id]	(FP algebra)
	$=$ eq0 → ~1; g @ [∗ @ [~1, id], sub1]	(FP algebra)
	$=$ eq0 → ~1; ∗ @ [id, g @ [~1, sub1]]	(10.12)
	$=$ eq0 → ~1; ∗ @ [id, newfact @ sub1]	(FP algebra)
	$=$ eq0 → ~1; ∗ @ [id, fact @ sub1]	(induction)
	$=$ fact	(definition). QED

So newfact is correct if we assume that fact is correct. This is a plausible assumption because fact is just a translation of the definition of the factorial function.

end example

Exercises

Relational Algebra

1. Given the following relations R and S with attribute sets $\{A, B, C, D\}$ and $\{B, C, D, E\}$, respectively.

R

A	*B*	*C*	*D*
1	*a*	#	*M*
2	*a*	*	*N*
1	*b*	#	*M*
3	*a*	%	*N*

S

B	*C*	*D*	*E*
a	#	*M*	*x*
b	*	*N*	*y*
a	#	*M*	*z*
b	%	*M*	*w*

Compute each of the following relations.

a. select(R, B, a).
b. project$(R, \{B, D\})$.
c. join(R, S).
d. project$(R, \{B, C, D\}) \cup$ project$(S, \{B, C, D\})$.
e. project$(R, \{B, D\}) \cap$ project$(S, \{B, D\})$.

2. Give an example of two relations R and S that have the same set of attributes $\{A, B\}$ and such that join$(R, S) \neq \varnothing$ and join$(R, S) \neq R \cup S$.

3. Find a relational algebra expression for each of the following questions about the example relations in Figures 10.11, 10.12, and 10.13.

a. Construct a relation consisting of cable channel stations. The expression should evaluate to {(AMC, 48), (CNN, 96), ...}.
b. What are the names of the movie channel stations? The expression should evaluate to {(AMC), (TCM), ...}.
c. Which rooms with computers have white boards too? The expresssion should evaluate to {(SC211), ... }.
d. How many seats are in CH301? The expression should evalutate to {(90)}.
e. What information exists about ESPN? The expression should evaluate to {(ESPN, 140, 32, Sports)}.
f. Is there a computer in the room where CS 252 Section 1 is taught? The expression should evaluate to {(Yes)}.

4. Use the definitions for the operators select and join to prove each of the following listed properties.

a. $\text{select}_{A=a}(\text{select}_{B=b}(R)) = \text{select}_{B=b}(\text{select}_{A=a}(R))$.
b. $R \bowtie R = R$.
c. $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$.

5. Let R be a relation, X a set of attributes of R, and A an attribute in X. Prove the following relationship between project and select.

$$\text{project}_X(\text{select}_{A=a}(R)) = \text{select}_{A=a}(\text{project}_X(R)).$$

Functional Algebra

6. Write an FP function to implement each of the following definitions.

a. $f(n) = \langle(0, 0), (1, 1), \ldots, (n, n)\rangle$.
b. $f((x_1, \ldots, x_n), (y_1, \ldots, y_n)) = \langle(x_1, y_1), \ldots, (x_n, y_n)\rangle$.

7. Prove each of the following FP equations.

a. $+ @ [1, 2] = + @ [2, 1] = +$.
b. $1 @ \sim\langle a, b\rangle = \sim a$ and $2 @ \sim\langle a, b\rangle = \sim b$.

8. Prove the following FP equation (10.12) from Example 10.30.

$$* @ [a, g @ [b, c]] = g @ [* @ [a, b], c],$$

where a, b, and c are any functions that return natural numbers and g has the following definition.

$$g = \text{eq0} @ 2 \rightarrow 1;\ g @ [*, \text{sub1} @ 2].$$

9. The following FP function is a translation of the recursive definition for the nth Fibonacci number, where sub2 is the FP function to subtract 2:

$$\text{slow} = \text{eq0} \rightarrow \sim 0;\ \text{eq1} \rightarrow \sim 1;\ + \ @\ [\text{slow} \ @\ \text{sub1},\ \text{slow} \ @\ \text{sub2}].$$

The following FP function claims to compute the nth Fibonacci number by iteration:

$$\text{fast} = 1 \ @\ g, \quad \text{where} \quad g = \text{eq0} \rightarrow \sim \langle 0,\ 1\rangle;\ [2\ ,\ +]\ @\ g\ @\ \text{sub1}.$$

Prove that slow and fast are equivalent FP functions.

10.5 Other Algebraic Ideas

In this section we introduce some general algebraic tools that are used to solve some computational problems. We'll introduce congruence mod n as a computational tool and after studying some simple properties we'll see how they apply to computations in cryptology.

We'll also give a short description of some tools and techniques for constructing new algebras from existing algebras. We'll describe subalgebras, which have applications to defining new abstract data types from existing ones. Then we'll describe morphisms, which are used to transform one object into another without destroying certain properties.

10.5.1 Congruence

We're going to examine a useful property that sometimes occurs when an equivalence relation interacts with algebraic operations in a certain way. We'll use the familiar mod function that we discussed in Chapter 2.

Recall that if x is an integer and n is a positive integer, then x mod n denotes the remainder upon division of x by n. We can define an equivalence relation on the integers by relating two numbers x and y if the following equation holds:

$$x \bmod n = y \bmod n.$$

For example, if $n = 4$, then the integers are partitioned into the following four equivalence classes, where $[k] = \{x \mid x \bmod 4 = k \bmod 4\}$.

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

We'll soon see that we can consider such classes to be elements of an algebra. Before we do this we'll introduce a notational convenience. Since the equation x mod $n = y$ mod n is used repeatedly when dealing with numbers, the following notation has been developed.

$$x \equiv y \pmod{n}.$$

We say that x is *congruent* to y mod n. Remember, it still just means the same old thing that $x - y$ is divisible by n.

Now, we can notice two very interesting arithmetic properties of the mod function and how it interacts with addition and multiplication.

Two Properties of the Mod Function. **(10.13)**

If $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then

$$a + c \equiv b + d \pmod{n} \quad \text{and} \quad ac \equiv bd \pmod{n}.$$

Proof: The hypotheses tell us that there are integers k and l such that

$$a = b + kn \quad \text{and} \quad c = d + ln.$$

Adding the two equations, we get

$$a + c = b + d + (k + l)n.$$

so that $a + c \equiv b + d \pmod{n}$. Now multiply the two equations to get

$$ac = bd + (bl + kd + kln)n,$$

which gives $ac \equiv bd \pmod{n}$. QED.

The two properties (10.13) are an example of operations that interact with an equivalence relation in a special way, which we'll now describe. Suppose $\sim$ is an equivalence relation on a set A. An n-ary operation f on A is said to *preserve* $\sim$ if it satisfies the following property:

$$\text{If } a_1 \sim b_1, \ldots, a_n \sim b_n, \text{ then } f(a_1, \ldots, a_n) \sim f(b_1, \ldots, b_n).$$

For example, (10.13) says that the mod n relation is preserved by addition and multiplication.

When an equivalence relation on the carrier of an algebra is preserved by each operation of the algebra, then the relation is called a *congruence relation* on the algebra, and the expression $x \sim y$ is called a *congruence*. For example, the mod n relation is a congruence relation on the algebra $\langle \mathbb{Z}; +, \cdot \rangle$.

A Finite Algebra: The Integers Mod n

The two properties (10.13) allow us to think of the equivalence classes

$$[0],\ [1],\ \ldots,\ [n-1]$$

as the elements of an algebra where we can add and multiply them with the following definition:

$$[a] + [b] = [a + b] \quad \text{and} \quad [a]\cdot[b] = [a\cdot b].$$

We should note that these definitions make sense. In other words, if $[a] = [c]$ and $[b] = [d]$, then we must show that $[a + b] = [c + d]$ and $[a\cdot b] = [c\cdot d]$. Since $[a] = [c]$ and $[b] = [d]$, it follows that $a \equiv c \pmod n$ and $b \equiv d \pmod n$. So (10.13) tells us that $a + b \equiv c + d \pmod n$ and $a\cdot b \equiv c\cdot d \pmod n$ In other words, $[a + b] = [c + d]$ and $[a\cdot b] = [c\cdot d]$. So addition and multiplication of equivalence classes are indeed valid operations.

example **10.31 A Finite Algebra**

The mod 4 equivalence relation partitions the integers into the four classes [0], [1], [2], and [3]. These four classes are elements of an algebra. Here are a few sample calculations.

$$\begin{aligned}
&[2] + [3] = [2 + 3] = [5] = [1]\,,\\
&[2]\cdot[3] = [2\cdot 3] = [6] = [2]\,,\\
&[0] + [3] = [0 + 3] = [3]\,,\\
&[1]\cdot[2] = [1\cdot 2] = [2]\,.
\end{aligned}$$

We'll leave it as an exercise to write the addition and multiplication tables.

end example

Fermat's Little Theorem

We'll use the previous results to prove an old and quite useful result about numbers that is due to Fermat and is often called Fermat's little theorem.

Fermat's Little Theorem

If p is prime and a is not divisible by p, then $a^{p-1} \equiv 1 \pmod p$.

Proof: Let $[0], [1], \ldots, [p-1]$ denote the equivalence classes that partition the integers with respect to the mod p relation. Since a is not divisible by p, it follows that $[a] \neq [0]$ because $[0]$ is the set of all multiples of p (i.e., the set of all integers divisible by p). Now look at the sequence of classes

$$[1 \cdot a], [2 \cdot a], \ldots, [(p-1) \cdot a]$$

We can observe that each of these classes is not $[0]$. For example, if we had $[i \cdot a] = [0]$, this would imply that $i \cdot a$ is divisible by p. But p is relatively prime to a, so by (2.2d) it would have to divide i. But $i < p$, so i can't be divided by p. Therefore, $[i \cdot a] \neq [0]$. We can also observe that the classes are distinct. For example, if $[i \cdot a] = [j \cdot a]$, then we would have $i \cdot a \bmod p = j \cdot a \bmod p$. Since $\gcd(a, p) = 1$, it follows by (2.4d) that $i \bmod p = j \bmod p$, which tells us that $[i] = [j]$.

Since the classes $[1 \cdot a], [2 \cdot a], \ldots, [(p-1) \cdot a]$ are all distinct and not equal to $[0]$, they must be an arrangement of the sets $[1], [2], \ldots, [p-1]$. So the product of the classes $[1], [2], \ldots, [p-1]$ must be equal to the product of the classes $[1 \cdot a], [2 \cdot a], \ldots, [(p-1) \cdot a]$. In other words we have the following equality:

$$[1 \cdot a] \cdot [2 \cdot a] \cdot \cdots \cdot [(p-1) \cdot a] = [1] \cdot [2] \cdot \cdots \cdot [p-1].$$

Since $[x \cdot y] = [x] \cdot [y]$, we can rewrite this equation to obtain

$$[1 \cdot a \cdot 2 \cdot a \cdot \cdots \cdot (p-1) \cdot a] = [1 \cdot 2 \cdot \cdots \cdot (p-1)].$$

Put all the a's together to obtain the following equation.

$$[a^{p-1} \cdot 1 \cdot 2 \cdot \cdots \cdot (p-1)] = [1 \cdot 2 \cdot \cdots \cdot (p-1)]$$

This class equation means that

$$a^{p-1} \cdot 1 \cdot 2 \cdot \cdots \cdot (p-1) \bmod p = 1 \cdot 2 \cdot \cdots \cdot (p-1) \bmod p.$$

Since each of the numbers $1, 2, \ldots, p-1$ is relatively prime to p, it follows from (2.4d) that they can each be cancelled from the equation to obtain the following equation:

$$a^{p-1} \bmod p = 1 \bmod p = 1.$$

In other words, we have $a^{p-1} \equiv 1 \pmod{p}$, which is the desired result. QED.

10.5.2 Cryptology: The RSA Algorithm

In cryptology a *public-key cryptosystem* is a system for encrypting and decrypting messages in which the public is aware of the key that is needed to encrypt messages sent to the receiver. But the receiver is the only one who knows the private key needed to decrypt a message.

The first working system to accomplish this task is called the RSA algorithm, named after its founders Ronald Rivest, Adi Shamir, and Leonard Adleman [1978].

We'll describe the general idea of how the algorithm is used. Then we'll discuss the implementation and why it works. The algorithm works in the following way, where the message to be sent is a number. (This is no problem because any text can be transformed into a number in many ways.)

1. The receiver constructs a public encryption key, which is a pair of numbers (e, n), and makes it available to the public. The receiver also constructs a private decryption key d that no one else knows.

2. Any person with the public key (e, n) can send a message a to the receiver if $0 \le a < n$. The sender encrypts a to a number c with the following calculation.

$$c = a^e \bmod n.$$

   The sender then sends c to the receiver.

3. The receiver upon receiving c makes the following calculation to decrypt c, where d is the private key and n is taken from the public key (e, n).

$$c^d \bmod n.$$

   This value is the desired message a.

The Details of the Keys

To construct the keys, choose two large distinct prime numbers p and q and let $n = pq$. Then choose a positive integer d that is relatively prime to the product $(p - 1)(q - 1)$. Then let e be a positive integer that satisfies the equation $ed \bmod ((p - 1)(q - 1)) = 1$.

If the keys are constructed in this way, then the system works. In other words, we have the following theorem.

RSA Theorem

If $0 \le a < n$ and $c = a^e \bmod n$, then $c^d \bmod n = a$.

The proof depends on Fermat's little theorem. We'll give it in stages, beginning with two lemmas.

Lemma 1: $a^{ed} \bmod p = a \bmod p$ and $a^{ed} \bmod q = a \bmod q$.

Proof: The numbers e and d were chosen so that $ed \bmod (p - 1)(q - 1) = 1$. So we can write $ed = 1 + (p - 1)(q - 1)k$ for some integer k. We'll start with the following equations:

$$\begin{aligned} a^{ed} \bmod p &= a^{1+(p-1)(q-1)k} \bmod p \\ &= \left(a^1 \bmod p\right)\left(a^{(p-1)(q-1)k} \bmod p\right) \bmod p && (2.4c) \\ &= (a \bmod p)\left(a^{p-1} \bmod p\right)^{(q-1)k} \bmod p && (2.4c) \end{aligned}$$

Now we consider two cases. If $\gcd(a, p) = 1$, then we can apply Fermat's little theorem to obtain $a^{p-1} \bmod p = 1$. So the equations continue as follows:

$$\begin{aligned} &= (a \bmod p)\,(1)^{(q-1)k} \bmod p \\ &= a \bmod p. \end{aligned}$$

If $\gcd(a, p) \neq 1$, then, since p is prime, it must be the case that $\gcd(a, p) = p$ so that p divides a. In this case $a \bmod p = 0$ and thus also $a^{ed} \bmod p = 0$. So in either case, we obtain the desired result $a^{ed} \bmod p = a \bmod p$. A similar argument shows that $a^{ed} \bmod q = a \bmod q$. QED.

Lemma 2: $a^{ed} \bmod n = a$ for $0 \leq a < n$.

Proof: By Lemma 1 we know that $a^{ed} \bmod p = a \bmod p$. So p divides $a^{ed} - a$. In other words, $a^{ed} - a = pk$ for some integer k. But Lemma 1 also tells us that $a^{ed} \bmod q = a \bmod q$. So q divides $a^{ed} - a = pk$. Since p and q are distinct primes, we have $\gcd(p, q) = 1$. So q divides k. It follows that $k = ql$ for some integer l. Thus $a^{ed} - a = pql$ and it follows that pq divides $a^{ed} - a$. Therefore, $a^{ed} \bmod pq = a \bmod pq = a$ because $a < n = pq$. QED.

Proof of RSA Theorem:

The proof of the RSA Theorem is now a simple obsevation based on Lemma 2. Let $c = a^e \bmod n$. We must show that $c^d \bmod n = a$. The following sequence of equations does the job.

$$\begin{aligned} c^d \bmod n &= (a^e \bmod n)^d \bmod n \\ &= \left(a^{ed} \bmod n\right) \\ &= a. \text{ QED.} \end{aligned}$$

Practical Use of the RSA Algorithm

The practical use of the RSA algorithm is based on several pieces of mathematical knowledge. The security of the system is based on the fact that factoring large numbers is a very hard problem. If n is chosen to be the product of two very large prime numbers, then it will be very hard for someone to factor n to find the two prime numbers. So it will be very hard to construct the private decryption key d from the public encryption key (e, n).

The speed of the system is based on the fact that it is very easy to encrypt and decrypt numbers. This follows from some basic results about the mod function. For example, the RSA paper [1978] includes a simple algorithm to calculate $a^e \bmod n$ that requires at most $2 \cdot \log_2 e$ multiplications and $2 \cdot \log_2 e$ divisions. The calculation of $c^d \bmod n$ is similar. The paper also discusses fast methods to construct the keys in the first place: to find large prime numbers p and q; to construct the product $(p - 1)(q - 1)$; to choose d; and to compute e.

It's time to do some examples.

example

10.32 Generating the Keys

We'll construct keys for a simple example to see the construction method. Let $p = 13$ and $q = 17$. Then $n = pq = 221$ and $(p - 1)(q - 1) = 12 \cdot 16 = 192$. For the private key we'll choose $d = 19$, which is a prime number larger than either p or q, so we know it is relatively prime to $(p - 1)(q - 1) = 192$. To construct e, we must satisfy the equation ed mod $((p - 1)(q - 1)) = 1$, which becomes $19e$ mod $221 = 1$. Since gcd(19, 221) = 1, we can use Euclid's algorithm in reverse to find two numbers e and s such that $1 = 19e + 221s$. For example, to compute gcd(19, 221) we proceed as follows, where each equation has the form $a = b \cdot q + r$ with $0 \leq r < |b|$:

$$\begin{aligned}
221 &= 19 \cdot 11 + 12 \\
19 &= 12 \cdot 1 + 7 \\
12 &= 7 \cdot 1 + 5 \\
7 &= 5 \cdot 1 + 2 \\
5 &= 2 \cdot 2 + 1
\end{aligned}$$

The gcd is the last nonzero remainder, in this case 1. So we start with the remainder 1 in the fifth equation and work backwards eliminating the remainders 2, 5, 7, and 12.

$$\begin{aligned}
1 &= 5 - 2 \cdot 2 \\
&= 5 - (7 - 5) \cdot 2 \\
&= 5 \cdot 3 - 7 \cdot 2 \\
&= (12 - 7) \cdot 3 - 7 \cdot 2 \\
&= 12 \cdot 3 - 7 \cdot 5 \\
&= 12 \cdot 3 - (19 - 12) \cdot 5 \\
&= 12 \cdot 8 - 19 \cdot 5 \\
&= (221 - 19 \cdot 11) \cdot 8 - 19 \cdot 5 \\
&= 221 \cdot 8 + 19 \cdot (-93)\,.
\end{aligned}$$

Therefore, $1 = 19 \cdot (-93)$ mod 221. But we can't choose e to be -93 because e must be positive. This is no problem because we can add any multiple of 221 to -93. For example, let $e = 221 - 93 = 128$. We'll verify that this value of e works.

$$\begin{aligned}
de \bmod 221 &= 19 \cdot 128 \bmod 221 \\
&= 19 \cdot (221 - 93) \bmod 221 \\
&= ((19 \cdot 221 \bmod 221) + (19 \cdot (-93) \bmod 221)) \bmod 221 \\
&= (0 + 1) \bmod 221 \\
&= 1.
\end{aligned}$$

Therefore, the public key is $(e, n) = (128, 221)$ and the private key d is 19.

end example

example

10.33 Sending and Receiving a Message

We'll use the RSA algorithm to encrypt and decrypt a message. But first we need to agree on a way to represent the letters in the message. To keep things simple we'll identify the uppercase letters A, B, ..., Z with the integers 1, 2, ..., 26 and the blank space with 0. So each symbol can be represented by a 2-digit number. In other words, A = 01, B = 02, ..., Z = 26, and blank = 00.

To keep numbers a reasonable size, we'll break each message into a sequence of two-letter blocks. For example, to send the message HELLO WORLD we'll break it up into the following six 2-letter blocks.

HE = 0805, LL = 1212, O = 1500, WO = 2315, RL = 1812, D = 0400.

The largest number for any two letter block is ZZ = 2626. So we'll have to construct a public key (e, n), where $n > 2626$. We'll let $n = 2773 = 47 \cdot 59$, which is the smallest product of two primes that is greater then 2626. For this choice of n, an example in the RSA paper [1978] chose $d = 157$ and then computed $e = 17$. We'll use these values too.

For example, to encrypt a number x, we must calulate

$$x^{17} \bmod 2773.$$

For example, we'll use (2.4c) to encrypt the first block HE = 0805 as follows:

$$\begin{aligned}
805^{17} \bmod 2773 &= [(805)((((805)^2)^2)^2)^2] \bmod 2773 \\
&= [(805)(((805^2 \bmod 2773)^2)^2)^2] \bmod 2773 \\
&= [(805)(((1916)^2)^2)^2] \bmod 2773 \\
&= [(805)((1916^2 \bmod 2773)^2)^2] \bmod 2773 \\
&= [(805)((2377)^2)^2] \bmod 2773 \\
&= [(805)(2377^2 \bmod 2773)^2] \bmod 2773 \\
&= [(805)(1528)^2] \bmod 2773 \\
&= [(805)(1528^2 \bmod 2773)] \bmod 2773 \\
&= [(805)(2691)] \bmod 2773 \\
&= 542.
\end{aligned}$$

After encrypting all six blocks, we obtain the following six little messages to be sent out.

HE = 0542, LL = 2345, O = 2417, WO = 2639, RL = 2056, D = 0017.

We'll leave it as an exercise to decrypt this sequence of numbers into the original message. For example, $0542^{157} \bmod 2773 = 0805$.

end example

10.5.3 Subalgebras

Programmers often need to create new data types to represent information. Sometimes a new data type can use the same operations of an existing type. For example, suppose we have an integer type available to us but we need to detect an error condition whenever a negative integer is encountered. One way to solve the problem is to define a new type for the natural numbers that uses some of the operations of the integer type.

For example, we can still use +, ∗, mod, and div (integer divide) because $\mathbb{N}$ is "closed" with respect to these operations. In other words, these operations return values in $\mathbb{N}$ if their arguments are in $\mathbb{N}$. On the other hand, we can't use the subtraction operation because $\mathbb{N}$ isn't closed with respect to it (e.g., $3 - 4 \notin \mathbb{N}$). We say that our new type "inherits" the operations +, ∗, mod, and div from the existing integer type. In algebraic terms we've created a new algebra

$$\langle \mathbb{N}; +, *, \text{mod}, \text{div} \rangle,$$

which is a "subalgebra" of $\langle \mathbb{Z}; +, *, \text{mod}, \text{div} \rangle$.

Let's describe the general idea of a subalgebra. Let A be the carrier of an algebra, and let B be a subset of A. We say that B is *closed* with respect to an operation if the operation returns a value in B whenever its arguments are from B. The diagram in Figure 10.14 gives a graphical picture showing B is closed with respect to the binary operation $\circ$.

Definition of Subalgebra

If A is the carrier of an algebra and B is a subset of A that is closed with respect to all the operations of A, then B is the carrier of an algebra called a *subalgebra* of the algebra of A. In other words, if $\langle A; \Omega \rangle$ is an algebra, where Ω is the set of operations, and if B is a subset of A that is closed with respect to all operations in Ω, then $\langle B; \Omega \rangle$ is an algebra, called a *subalgebra* of $\langle A; \Omega \rangle$.

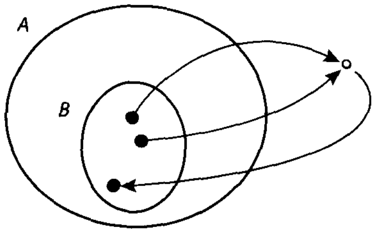


Figure 10.14 B is closed with respect to $\circ$.

example **10.34 Some Sample Subalgebras**

Here are three examples of subalgebras.

1. $\langle \mathbb{N}; +, *, \text{mod}, \text{div}\rangle$ is a subalgebra of $\langle \mathbb{Z}; +, *, \text{mod}, \text{div}\rangle$.
2. Let $\Omega = \{+, -, \cdot, 0, 1\}$. Then we have a sequence of subalgebras, where each one is a subalgebra of the next: $\langle \mathbb{Z}; \Omega\rangle$, $\langle \mathbb{Q}; \Omega\rangle$, $\langle \mathbb{R}; \Omega\rangle$.
3. Consider the algebra $\langle \mathbb{N}_8; +_8, 0\rangle$, where $+_8$ means addition mod 8. The set $\{0, 2, 4, 6\}$ forms the carrier of a subalgebra. But $\{0, 3, 6\}$ is not the carrier of a subalgebra because $3 +_8 6 = 1$ and $1 \notin \{0, 3, 6\}$.

end example

Combining Subalgebras

We can combine subalgebras by forming the intersection of the carriers. For example, consider the algebra $\langle \mathbb{N}_{12}; +_{12}, 0\rangle$, where $+_{12}$ means addition mod 12. Two subalgebras of this algebra have carriers $\{0, 2, 4, 6, 8, 10\}$ and $\{0, 3, 6, 9\}$. The intersection of these two carriers is the set $\{0, 6\}$, which forms the carrier of another subalgebra of $\langle \mathbb{N}_{12}; +_{12}, 0\rangle$. This is no fluke. It follows because the carrier of each subalgebra is closed with respect to the operations. So it follows that the intersection of carriers is also closed.

Generating a Subalgebra

One way to generate a new subalgebra is to take any subset you like—say, S—from the carrier of an algebra. If the operations of the algebra are closed with respect to S, then we have a new subalgebra. If not, then keep applying the operations of the algebra to elements of S. If an operation gives a result x and $x \notin S$, then enlarge S by adding x to form the bigger set $S \cup \{x\}$. Each time a bigger set is constructed, the process must start over again until the set is closed under the operations of the algebra. The resulting subalgebra has the smallest carrier that contains S.

example **10.35 Smallest Subalgebra**

We'll start with the algebra $\langle \mathbb{N}_{12}; +_{12}, 0\rangle$ and try to find the smallest subalgebra whose carrier contains the subset $\{4, 10\}$ of $\mathbb{N}_{12}$. Notice that this set is not closed under the operation $+_{12}$ because $4 +_{12} 4 = 8$, and $8 \notin \{4, 10\}$. So we'll add the number 8 to get the new subset $\{4, 8, 10\}$. Still there are problems because $4 +_{12} 8 = 0$, and $8 +_{12} 10 = 6$. So we'll add 0 and 6 to our set to obtain the subset $\{0, 4, 6, 8, 10\}$. We aren't done yet, because $6 +_{12} 8 = 2$. After adding 2, we obtain the set $\{0, 2, 4, 6, 8, 10\}$, which is closed under the operation of $+_{12}$ and contains the constant 0.

Therefore, the algebra $\langle\{0, 2, 4, 6, 8, 10\}; +_{12}, 0\rangle$ is the smallest subalgebra of $\langle\mathbb{N}_{12}; +_{12}, 0\rangle$ that contains the set $\{4, 10\}$.

end example

10.5.4 Morphisms

This little discussion is about some tools and techniques that can be used to compare two different entities for common properties. For example, if A is an alphabet, then we know that a string over A is different from a list over A. In other words, we know that A^* and lists(A) contain different kinds of objects. But we also know that A^* and lists(A) have a lot in common. For example, we know that the operations on A^* are similar to the operations on lists(A). We know that the algebra of strings and the algebra of lists both have an empty object and that they construct new objects in a similar way. In fact, we know that strings can be represented by lists.

On the other hand, we know that A^* is quite different from the set of binary trees over A. For example, the construction of a string is not at all like the construction of a binary tree.

We would like to be able to decide whether two different entities are alike in some way. When two things are alike, we are often more familiar with one of the things. So we can apply our knowledge about the familiar one and learn something about the unfamiliar one. This is a bit vague. So let's start off with a general problem of computer science:

The Transformation Problem

Transform an object into another object with some particular property.

This is a very general statement. So let's look at a few interpretations. For example, we may want the transformed object to be "simpler" than the original object. This usually means that the new object has the same meaning as the given object but uses fewer symbols. For example, the expression $x + 1$ might be a simplification of $(x^2 + x)/x$, and the FP program f @ (true $\rightarrow c$; d) can be simplified to f @ c.

We may want the transformed object to act as the meaning of the given object. For example, we usually think of the meaning of the expression $3 + 4$ as its value, which is 7. On the other hand, the meaning of the expression $x + 1$ is $x + 1$ if we don't know the value of x.

Whenever a light bulb goes on in our brain and we finally understand the meaning of some idea or object, we usually make statements like "Oh yes, I see it now" or "Yes, I understand." These statements usually mean that we have made a connection between the thing we're trying to understand and some other thing that is already familiar to us. So there is a transformation (i.e., a function) from the new idea to a familiar old idea.

Introductory Example: Sematics of Numerals

Suppose we want to describe the meaning of the base 10 numerals (i.e., nonempty strings of decimal digits) or the base 2 numerals (i.e., nonempty strings of binary digits). Let m_{ten} denote the meaning function for base 10 numerals, and let m_{two} denote the meaning function for base 2 numerals. If we can agree on anything, we most probably will agree that $m_{\text{ten}}(16) = m_{\text{two}}(10000)$ and $m_{\text{ten}}(14) = m_{\text{two}}(1110)$. Further, if we let m_{rom} denote the meaning function for Roman numerals, then we most probably also agree that $m_{\text{rom}}(\text{XII}) = m_{\text{ten}}(12) = m_{\text{two}}(1100)$.

For this example we'll use the set $\mathbb{N}$ of natural numbers to represent the meanings of the numerals. For base 10 and base 2 numerals there may be some confusion because, for example, the string 25 denotes a base 10 numeral and it also represents the natural number that we call 25. Given that this confusion exists, we have

$$m_{\text{ten}}(25) = m_{\text{two}}(11001) = m_{\text{rom}}(\text{XXV}) = 25.$$

So we can write down three functions from the three kinds of numerals (the syntax) to natural numbers (the semantics):

$$\begin{aligned} m_{\text{ten}} :&\quad \text{DecimalNumerals} \to \mathbb{N},\\ m_{two} :&\quad \text{BinaryNumerals} \to \mathbb{N},\\ m_{rom} :&\quad \text{RomanNumerals} \to \mathbb{N}. \end{aligned}$$

Can we give definitions of these functions? Sure. For example, a natural definition for m_{ten} can be given as follows: If $d_k d_{k-1} \ldots d_1 d_0$ is a base 10 numeral, then

$$m_{\text{ten}}(d_k d_{k-1} \ldots d_1 d_0) = 10^k d_k + 10^{k-1} d_{k-1} + \cdots + 10 d_1 + d_0.$$

Preserving Operations

What properties, if any, should a semantics function possess? Certain operations defined on numerals should be, in some sense, "preserved" by the semantics function. For example, suppose we let $+_{bi}$ denote the usual binary addition defined on binary numerals. We would like to say that the meaning of the binary sum of two binary numerals is the same as the result obtained by adding the two individual meanings in the algebra $\langle \mathbb{N}; + \rangle$. In other words, for any binary numerals x and y, the following equation holds:

$$m_{\text{two}}(x +_{bi} y) = m_{\text{two}}(x) + m_{\text{two}}(y).$$

The idea of a function preserving an operation can be defined in a general way. Let $f : A \to A'$ be a function between the carriers of two algebras. Suppose ω is an n-ary operation on A. We say that f *preserves* the operation ω if there

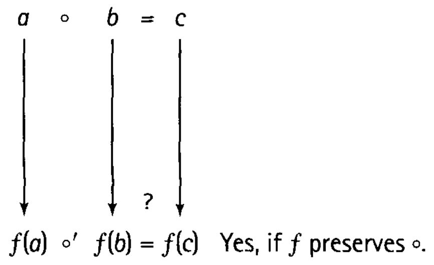


Figure 10.15 Preserving a binary operation.

is a corresponding operation ω' on A' such that, for every $x_1, \ldots, x_n \in A$, the following equality holds:

$$f(\omega\ (x_1, \ldots, x_n)) = \omega'(f(x_1), \ldots, f(x_n)).$$

Of course, if ω is a binary operation, then we can write the above equation in its infix form as follows:

$$f(x\ \omega\ y) = f(x)\ \omega'\ f(y).$$

For example, the binary numeral meaning function m_{two} preserves $+_{bi}$. We can write the equation using the prefix form of $+_{bi}$ as follows:

$$m_{\text{two}}(+_{bi}(x,\ y)) = +\ (m_{\text{two}}(x),\ m_{\text{two}}(y)).$$

Here's the thing to remember about an operation that is preserved by a function $f : A \to A'$: You can apply the operation to arguments in A and then use f to map the result to A', or you can use f to map each argument from A to A' and then apply the corresponding operation on A' to these arguments. In either case you get the same result.

Figure 10.15 illustrates this property for two binary operators $\circ$ and $\circ'$. In other words, if $a \circ b = c$ in A, then $f(a \circ b) = f(c) = f(a) \circ' f(b)$ in A'.

Definition of Morphism

We say that $f : A \to A'$ is a *morphism* (also called a *homomorphism*) if every operation in the algebra of A is preserved by f. If a morphism is injective, then it's called a *monomorphism*. If a morphism is surjective, then it's called an *epimorphism*. If a morphism is bijective, then it's called an *isomorphism*. If there is an isomorphism between two algebras, we say that the algebras are *isomorphic*. Two isomorphic algebras are very much alike, and, hopefully, one of them is easier to understand.

For example, m_{two} is a morphism from $\langle\text{BinaryNumerals}; +_{bi}\rangle$ to $\langle\mathbb{N}; +\rangle$. In fact, we can say that m_{two} is an epimorphism because it's surjective. Notice that distinct binary numerals like 011 and 11 both represent the number 3. Therefore, m_{two} is not injective, so it is not a monomorphism, and thus it is not an isomorphism.

example

10.36 A Morphism

Suppose we define $f : \mathbb{Z} \to \mathbb{Q}$ by $f(n) = 2^n$. Notice that

$$f(n + m) = 2^{n+m} = 2^n \cdot 2^m = f(n) \cdot f(m).$$

So f is a morphism from the algebra $\langle \mathbb{Z}; + \rangle$ to the algebra $\langle \mathbb{Q}; \cdot \rangle$. Notice that $f(0) = 2^0 = 1$. So f is a morphism from the algebra $\langle \mathbb{Z}; +, 0 \rangle$ to the algebra $\langle \mathbb{Q}; \cdot, 1 \rangle$. Notice that $f(-n) = 2^{-n} = (2^n)^{-1} = f(n)^{-1}$. Therefore, f is a morphism from the algebra $\langle \mathbb{Z}; +, -, 0 \rangle$ to the algebra $\langle \mathbb{Q}; \cdot, ^{-1}, 1 \rangle$. It's easy to see that f is injective and that f is not surjective. Therefore, f is a monomorphism, but it is neither an epimorphism nor an isomorphism.

end example

example

10.37 The Mod Function

Let $m > 1$ be a natural number, and let the function $f : \mathbb{N} \to \mathbb{N}_m$ be defined by $f(x) = x \bmod m$. We'll show that f is a morphism from $\langle \mathbb{N}, +, \cdot, 0, 1 \rangle$ to the algebra $\langle \mathbb{N}_m, +_m, \cdot_m, 0, 1 \rangle$. For f to be a morphism we must have $f(0) = 0$, $f(1) = 1$, and for all $x, y \in \mathbb{N}$:

$$f(x + y) = f(x) +_m f(y) \text{ and } f(x \cdot y) = f(x) \cdot_m f(y).$$

It's clear that $f(0) = 0$ and $f(1) = 1$. The other equations are just restatements of the congruences (10.13).

end example

example

10.38 Strings and Lists

For any alphabet A we can define a function $f : A^* \to \text{lists}(A)$ by mapping any string to the list consisting of all letters in the string. For example, $f(\Lambda) = \langle\ \rangle$, $f(a) = \langle a \rangle$, and $f(aba) = \langle a, b, a \rangle$. We can give a formal definition of f as follows:

$$f(\Lambda) = \langle\ \rangle,$$
$$f(a \cdot t) = a :: f(t) \text{ for every } a \in A \text{ and } t \in A^*.$$

For example, if $a \in A$, then $f(a) = f(a \cdot \Lambda) = a :: f(\Lambda) = a :: \langle\ \rangle = \langle a \rangle$. It's easy to see that f is bijective because any two distinct strings get mapped to two distinct lists and that any list is the image of some string.

We'll show that f preserves the concatenation of strings. Let "cat" denote both the concatenation of strings and the concatenation of lists. Then we must verify that $f(\text{cat}(s, t)) = \text{cat}(f(s), f(t))$ for any two strings s and t. We'll do it by induction on the length of s. If $s = \Lambda$, then we have

$$f(\text{cat}(\Lambda, t)) = f(t) = \text{cat}(\langle\ \rangle, f(t)) = \text{cat}(f(\Lambda), f(t)).$$

Now assume that s has length $n > 0$ and $f(\text{cat}(u, t)) = \text{cat}(f(u), f(t))$ for all strings u of length less than n. Since the length of s is greater than 0, we can write $s = a \cdot x$ for some $a \in A$ and $x \in A^*$. Then we have

$$\begin{aligned} f(\text{cat}(a \cdot x, t)) &= f(a \cdot \text{cat}(x, t)) && \text{(definition of string cat)} \\ &= a :: f(\text{cat}(x, t)) && \text{(definition of } f\text{)} \\ &= a :: \text{cat}(f(x), f(t)) && \text{(induction assumption)} \\ &= \text{cat}(a :: f(x), f(t)) && \text{(definition of list cat)} \\ &= \text{cat}(f(a \cdot x), f(t)) && \text{(definition of } f\text{).} \end{aligned}$$

Therefore, f preserves concatenation. Thus f is a morphism from the algebra $\langle A^*;\ \text{cat},\ \Lambda\rangle$ to the algebra $\langle \text{lists}(A);\ \text{cat},\ \langle\ \rangle\rangle$. Since f is also a bijection, it follows that the two algebras are isomorphic.

end example

Constructing Morphisms

Now let's consider the problem of constructing a morphism. We'll demonstrate the ideas with an example. Suppose we need a function $f : \mathbb{N}_8 \to \mathbb{N}_8$ with the property that $f(1) = 3$ and also f must be a morphism from the algebra $\langle \mathbb{N}_8;\ +_8,\ 0\rangle$ to itself, where $+_8$ is the operation of addition mod 8. We'll finish the definition of f. For f to be a morphism it must preserve $+_8$ and 0. So we must set $f(0) = 0$. What value should we assign to $f(2)$? Notice that we can write $2 = 1 +_8 1$. Since $f(1) = 3$ and f must preserve the operation $+_8$, we can obtain the value $f(2)$ as follows:

$$f(2) = f(1 +_8 1) = f(1) +_8 f(1) = 3 +_8 3 = 6.$$

Now we can compute $f(3) = f(1 +_8 2) = f(1) +_8 f(2) = 3 +_8 6 = 1$. Continuing, we get the following values: $f(4) = 4$, $f(5) = 7$, $f(6) = 2$, and $f(7) = 5$. So the two facts $f(0) = 0$ and $f(1) = 3$ are sufficient to define f.

But does this definition of f result in a morphism? We must be sure that $f(x +_8 y) = f(x) +_8 f(y)$ for all $x, y \in \mathbb{N}_8$. For example, is $f(3 +_8 6) = f(3) +_8 f(6)$? We can check it out easily by computing the left- and right-hand sides of the equation:

$$f(3 +_8 6) = f(1) = 3 \quad \text{and} \quad f(3) +_8 f(6) = 1 +_8 2 = 3.$$

Do we have to check the function for all possible pairs (x, y)? No. Our method for defining f was to force the following equation to be true:

$$f(1 +_8 \cdots +_8 1) = f(1) +_8 \cdots +_8 f(1).$$

Since any number in $\mathbb{N}_8$ is a sum of 1's, we are assured that f is a morphism. Let's write this out for an example:

$$\begin{aligned} f(3 +_8 4) &= f(1 +_8 1 +_8 1 +_8 1 +_8 1 +_8 1 +_8 1) \\ &= f(1) +_8 f(1) +_8 f(1) +_8 f(1) +_8 f(1) +_8 f(1) +_8 f(1) \\ &= [f(1) +_8 f(1) +_8 f(1)] +_8 [f(1) +_8 f(1) +_8 f(1) +_8 f(1)] \\ &= f(1 +_8 1 +_8 1) +_8 f(1 +_8 1 +_8 1 +_8 1) \\ &= f(3) +_8 f(4). \end{aligned}$$

The above discussion might convince you that once we pick $f(1)$, then we know $f(x)$ for all x. But if the codomain is a different carrier, then things can break down. For example, suppose we want to define a morphism f from the algebra $\langle \mathbb{N}_3;\ +_3,\ 0\rangle$ to the algebra $\langle \mathbb{N}_6;\ +_6,\ 0\rangle$. Then we must have $f(0) = 0$. Now, suppose we try to set $f(1) = 3$. Then we must have $f(2) = 0$.

$$f(2) = f(1 +_3 1) = f(1) +_6 f(1) = 3 +_6 3 = 0.$$

Is this definition of f a morphism? The answer is No! Notice that $f(1 +_3 2) \neq f(1) +_6 f(2)$, because $f(1 +_3 2) = f(0) = 0$ and $f(1) +_6 f(2) = 3 +_6 0 = 3$. So morphisms are not as numerous as one might think.

example

10.39 Language Morphisms

If A and B are alphabets, then a function $f : A^* \to B^*$ is called a *language morphism* if $f(\Lambda) = \Lambda$ and $f(uv) = f(u)f(v)$ for any strings $u,\ v \in A^*$. In other words, a language morphism from A^* to B^* is a morphism from the algebra $\langle A^*;\ \text{cat},\ \Lambda\rangle$ to the algebra $\langle B^*;\ \text{cat},\ \Lambda\rangle$. Since concatenation must be preserved, a language morphism is completely determined by defining the values $f(a)$ for each $a \in A$.

For example, let $A = B = \{a,\ b\}$ and define $f : \{a,\ b\}^* \to \{a,\ b\}^*$ by setting $f(a) = b$ and $f(b) = ab$. Then we can make statements like

$$f(bab) = f(b)f(a)f(b) = abbab \quad \text{and} \quad f(b^2) = (ab)^2.$$

Language morphisms can be used to transform one language into another language with a similar grammar. For example, the grammar

$$S \to aSb \mid \Lambda$$

defines the language $\{a^n b^n \mid n \in \mathbb{N}\}$. Since $f(a^n b^n) = b^n(ab)^n$ for $n \in \mathbb{N}$, the set $\{a^n b^n \mid n \in \mathbb{N}\}$ is transformed by f into the set $\{b^n(ab)^n \mid n \in \mathbb{N}\}$. This language can be generated by the grammar $S \to f(a)Sf(b) \mid f(\Lambda)$, which becomes $S \to bSab \mid \Lambda$.

end example

example **10.40 Casting Out by Nines, Threes, etc.**

An old technique for finding some answers and checking errors in some arithmetic operations is called "casting out by nines." We want to study the technique and see why it works (so it's not magic). Is 44,820 divisible by 9? Is 43·768 + 9579 = 41593? We can use casting out by nines to answer yes to the first question and no to the second question. How does the idea work? It's a consequence of the following result:

Casting Out by Nines (10.14)

If K is a natural number with decimal representation $d_n \ldots d_0$, then

$$K \bmod 9 = (d_n \bmod 9) +_9 \cdots +_9 (d_0 \bmod 9).$$

Proof: For the two algebras $\langle \mathbb{N};\ +,\ \cdot,\ 0,\ 1\rangle$ and $\langle \mathbb{N}_9;\ +_9,\ \cdot_9,\ 0,\ 1\rangle$, the function $f\colon\ \mathbb{N} \to \mathbb{N}_9$ defined by $f(x) = x \bmod 9$ is a morphism. We can also observe that $f(10) = 1$, and in fact $f(10^n) = 1$ for any natural number n. Now, since $d_n \ldots d_0$ is the decimal representation of K, we can write

$$K = d_n \cdot 10^n + \cdots + d_1 \cdot 10 + d_0.$$

Now apply f to both sides of the equation to get the desired result.

$$\begin{aligned} f(K) &= f(d_n \cdot 10^n + \cdots + d_1 \cdot 10 + d_0) \\ &= f(d_n) \cdot_9 f(10^n) +_9 \cdots +_9 f(d_1) \cdot_9 f(10) +_9 f(d_0) \\ &= f(d_n) \cdot_9 1 +_9 \cdots +_9 f(d_1) \cdot_9 1 +_9 f(d_0) \\ &= f(d_n) +_9 \cdots +_9 f(d_1) +_9 f(d_0). \quad \text{QED} \end{aligned}$$

Casting out by nines works because 10 mod 9 = 1. Therefore, casting out by threes also works because 10 mod 3 = 1. In general, for a base B number system, casting out by the predecessor of B works if we have the equation

$$B \bmod \operatorname{pred}(B) = 1.$$

For example, in octal, casting out by sevens works. (Do any other numbers work in octal?) But in binary, casting out by ones does not work because 2 mod 1 = 0.

end example

Exercises

Congruences

1. The equivalence relation $x \equiv y \pmod 4$ partitions the integers into the four equivalence classes [0], [1], [2], [3]. Construct the addition and multiplication tables for these classes.

2. (*Chinese Remainder Theorem*). Given n congruences

$$x \equiv a_1 \pmod{m_1}, \ldots, x \equiv a_n \pmod{m_n},$$

where $\gcd(m_i, m_j) = 1$ for each $i \neq j$. The following algorithm finds a unique solution x such that $0 \leq x < m$, where $m = m_1 \cdots m_n$.

1. For each i find b_i such that $(m/m_i)b_i \equiv 1 \pmod{m_i}$.
2. Set $x = (m/m_1)b_1a_1 + \cdots + (m/m_n)b_na_n$.
3. If x is not in the proper range, then add or subtract a multiple of m.

Find the unique solution to each of the following sets of congruences.

a. $x \equiv 8 \pmod{13}$ $x \equiv 3 \pmod{8}$.	b. $x \equiv 34 \pmod{9}$ $x \equiv 23 \pmod{10}$.	c. $x \equiv 17 \pmod{6}$ $x \equiv 15 \pmod{11}$.
d. $x \equiv 1 \pmod{2}$ $x \equiv 2 \pmod{3}$. $x \equiv 1 \pmod{5}$.	e. $x \equiv 3 \pmod{2}$ $x \equiv 1 \pmod{5}$. $x \equiv 0 \pmod{7}$.	f. $x \equiv 12 \pmod{3}$ $x \equiv 4 \pmod{7}$. $x \equiv 15 \pmod{11}$.

3. Prove the following statement about integers: If $x < 0$, then there is some $y > 0$ such that $y \equiv x \pmod{n}$.

4. (*A Pigeonhole Proof*). An interesting result about integers states that if $\gcd(a, n) = 1$, then there is an integer k in the range $1 \leq k \leq n$ such that $a^k \equiv 1 \pmod{n}$. A proof of this fact starts out as follows: Consider the set of $n + 1$ numbers

$$a, a^2, a^3, \ldots, a^{n+1}.$$

Calculate the set of remainders of these numbers upon division by n. In other words, we have the set of numbers

$$a \bmod n,\ a^2 \bmod n,\ a^3 \bmod n,\ \ldots,\ a^{n+1} \bmod n.$$

These $n + 1$ numbers are all in the range 0 to $n - 1$. By the pigeonhole principle two of the numbers must be identical. So for some $i < j$ we have $a^i \bmod n = a^j \bmod n$—in other words, $a^i \equiv a^j \pmod{n}$. This means that n divides $a^j - a^i = a^i(a^{j-i} - 1)$. Finish the proof by using properties of mod and gcd.

Cryptology

5. For each of the following cases, verify that n and d satisfy the requirements of the RSA algorithm and construct an encryption key e.

a. $p = 5,\ q = 7,\ n = 35,\ d = 11$.

b. $p = 7,\ q = 11,\ n = 77,\ d = 13$.

c. $p = 47,\ q = 59,\ n = 2773,\ d = 101$.

6. Given the value $n = 47{\cdot}59 = 2773$ from Example 10.33.
   a. Verify that 17 is a valid choice for the encrypting key e.
   b. Verify that 157 is a valid choice for the decrypting key d.
   c. Verify the values of the encrypted numbers for HELLO WORLD.
   d. Decrypt the encrypted numbers to verify that they are the original six numbers for HELLO WORLD. *Hint:* The decryption key 157 can be written $157 = 1 + 4 + 8 + 16 + 128$. So for any number x we have $x^{157} \bmod 2773 = (x{\cdot}x^4{\cdot}x^8{\cdot}x^{16}{\cdot}x^{128}) \bmod 2773$.

Subalgebras

7. For each of the following sets, state whether the set is the carrier of a subalgebra of the algebra $\langle \mathbb{N}_9;\ +_9,\ 0\rangle$.
   a. $\{0, 3, 6\}$. b. $\{1, 4, 5\}$. c. $\{0, 2, 4, 6, 8\}$.

8. Given the algebra $\langle \mathbb{N}_{12};\ +_{12},\ 0\rangle$, find the carriers of the subalgebras generated by each of the following sets.
   a. $\{6\}$. b. $\{3\}$. c. $\{5\}$.

Morphisms

9. Find the three morphisms that exist from the algebra $\langle \mathbb{N}_3;\ +_3,\ 0\rangle$ to the algebra $\langle \mathbb{N}_6;\ +_6,\ 0\rangle$.

10. Let A be an alphabet and $f : A^* \to \mathbb{N}$ be defined by $f(x) = \text{length}(x)$. Show that f is a morphism from the algebra $\langle A^*;\ \text{cat},\ \Lambda\ \rangle$ to $\langle \mathbb{N};\ +,\ 0\rangle$, where cat denotes the concatenation of strings.

11. Give an example to show that the absolute value function abs : $\mathbb{Z} \to \mathbb{N}$ defined by $\text{abs}(x) = |x|$ is not a morphism from the algebra $\langle \mathbb{Z};\ +\rangle$ to the algebra $\langle \mathbb{N};\ +\rangle$.

12. Let's assume that we know that the operation $+_n$ is associative over $\mathbb{N}_n$. Let $\circ$ be the binary operation over $\{a, b, c\}$ defined by the following table:

$\circ$	a	b	c
a	c	a	b
b	a	b	c
c	b	c	a

Show that $\circ$ is associative by finding an isomorphism of the two algebras $\langle\{a, b, c\};\ \circ\rangle$ and $\langle \mathbb{N}_3;\ +_3\rangle$.

13. Given the language morphism $f : \{a, b\}^* \rightarrow \{a, b\}^*$ defined by $f(a) = b$ and $f(b) = ab$, compute the value of each of the following expressions.

    a. $f(\{b^n a \mid n \in \mathbb{N}\})$.
    b. $f(\{ba^n \mid n \in \mathbb{N}\})$.
    c. $f^{-1}(\{b^n a \mid n \in \mathbb{N}\})$.
    d. $f^{-1}(\{ba^n \mid n \in \mathbb{N}\})$.
    e. $f^{-1}(\{ab^{n+1} \mid n \in \mathbb{N}\})$.

10.6 Chapter Summary

An algebra consists of one or more sets, called carriers, together with operations on the sets. An algebra is useful for solving problems when we have a good knowledge of its operations. We can use the properties of the operations to transform algebraic expressions into equivalent simpler expressions. In high school algebra the carrier is the set of real numbers, and the operations are addition, multiplication, and so on.

An abstract algebra is described by giving a set of axioms to describe the properties of its operations. An abstract algebra is useful when it has lots of concrete examples. Two especially useful concrete examples of Boolean algebra are the algebra of sets and the algebra of propositions. Some important properties of Boolean algebra operations are the idempotent properties, the absorption laws, the involution law, and De Morgan's laws. Digital circuits are modeled by Boolean algebraic expressions. Thus Boolean algebra can be used to simplify a digital circuit by simplifying the corresponding algebraic expression.

The abstract data types of computer science can be described as algebras. When an abstract data type is described as an algebra, its operations can be implemented and then checked for correctness against the axioms. Some fundamental abstract data types are the natural numbers, lists, strings, stacks, queues, binary trees, and priority queues.

Two algebras that are useful as computational tools are relational algebras for databases and functional algebras for reasoning about functional programs.

Many other algebraic ideas are quite useful for computational problems. Congruences are useful for describing properties of numbers and there are direct applications to cryptology. Subalgebras can be used to define new abstract data types. Morphisms allow us to transform one algebra into another—often simpler—algebra and still preserve the meaning of the operations. Language morphisms can be used to generate new languages along with their grammars.

This is not the end. It is not even the beginning of the end.
But it is, perhaps, the end of the beginning.

—Winston Churchill (1874–1965)

Answers to Selected Exercises

Chapter 1

Section 1.1

1. Consider the four statements "if $1 = 1$ then $2 = 2$," "if $1 = 1$ then $2 = 3$," "if $1 = 0$ then $2 = 2$," and "if $1 = 0$ then $2 = 3$." The second statement is the only one that is false.

3. a. 47 is a prime between 45 and 54. **c.** The statement is true. **e.** The statment is false. For example, the numbers 2 and 5 have the desired form: $2 = 3(0) + 2$ and $5 = 3(1) + 2$. But the product $2(5) = 10$ can't be written as $10 = 3k + 2$ because the equation does not have an integer solution for k.

4. a. Let x and y be any two even integers. Then they can be written in the form $x = 2m$ and $y = 2n$ for some integers m and n. Therefore, the sum $x + y$ can be written as $x + y = 2m + 2n = 2(m + n)$, which is an even integer.
c. Let x and y be any odd integers. Then they can be written $x = 2m + 1$ and $y = 2n + 1$ for some integers m and n. Therefore, we have $x - y = 2m + 1 - 2n - 1 = 2(m - n)$, which is an even integer.

6. a. Let $x = 3m + 4$, and let $y = 3n + 4$ for some integers m and n. Then $xy = (3m + 4)(3n + 4) = 9mn + 12m + 12n + 16 = 3(3mn + 4m + 4n + 4) + 4$, which has the desired form. **c.** Let $x = 7m + 8$, and let $y = 7n + 8$ for some integers m and n. Then $xy = (7m + 8)(7n + 8) = 49mn + 56m + 56n + 64 = 7(7mn + 8m + 8n + 8) + 8$, which has the desired form.

7. a. Let $d \mid (da + b)$. Then $da + b = dk$ for some intger k. Solving the equation for b gives $b = d(k - a)$, which says that $d \mid b$.
c. Let $d \mid a$ and $d \mid b$. Then there are integers k and j such that $a = dk$ and $b = dj$. So for any integers x and y we have $ax + by = dkx + djy = d(kx + jy)$, which says that $d \mid (ax + by)$.

8. a. First we'll prove the statement "If x is even then x^2 is even." If x is even, then $x = 2n$ for some integer n. Therefore, $x^2 = (2n)(2n) = 2(2n^2)$, which is even. Next we'll prove the statement "If x^2 is even then x is even," by proving the contrapositive "If x is odd, then x^2 is odd." So if x is odd, then $x = 2n + 1$ for some integer n. Thus $x^2 = (2n + 1)(2n + 1) = 4n^2 + 4n + 1 = 2(2n^2 + 2n) + 1$, which is an odd integer. So the iff statement is proven.
c. $x^2 + 6x + 9$ is even iff $(x + 3)^2$ is even iff $(x + 3)$ is even iff x is odd.

Section 1.2

1. a. $\{1, 2, 3, 4, 5, 6, 7\}$. **c.** $\{3, 5, 7, 11, 13, 17, 19\}$.
e. M, I, S, P, R, V, E.

2. a. $\{x \mid x \in \mathbb{N} \text{ and } 1 \leq x \leq 31\}$.
c. $\{x \mid x = n^2 \text{ and } n \in \mathbb{N} \text{ and } 1 \leq n \leq 8\}$ or $\{x^2 \mid x \in \mathbb{N} \text{ and } 1 \leq x \leq 8\}$.

3. a. True. **c.** False. **e.** True. **g.** True.

5. For example, let $A = \{x\}$ and $B = \{x, \{x\}\}$.

6. a. $\{\varnothing, \{x\}, \{y\}, \{z\}, \{w\}, \{x, y\}, \{x, z\}, \{x, w\}, \{y, z\}, \{y, w\}, \{z, w\}, \{x, y, z\}, \{x, y, w\}, \{x, z, w\}, \{y, z, w\}, \{x, y, z, w\}\}$. **c.** $\{\varnothing\}$. **e.** $\{\varnothing, \{\{a\}\}, \{\varnothing\}, \{\{a\}, \varnothing\}\}$.

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

8. a. $A \cup B = \{1, 5, 8, 9, 11, 13, 14, 17, 20, 21, \ldots\}$.

9. No. A counterexample is $A = \{a\}$ and $B = \{b\}$.

10. a. $A_0 = \mathbb{Z} - \{0\}$, $A_1 = \{0\}$, $A_2 = \mathbb{Z} - \{-2, -1, 0, 1, 2\}$, $A_3 = \{-2, -1, 0, 1, 2\}$, $A_{-2} = \mathbb{Z}$ and $A_{-3} = \varnothing$. **c.** $\mathbb{Z}$. **e.** $\mathbb{Z}$. **g.** $\varnothing$. **i.** $\varnothing$.

11. a. $A_0 = \mathbb{N} - \{0\}$, $A_1 = \{1\}$, $A_2 = \{1, 2\}$, $A_3 = \{1, 3\}$, $A_4 = \{1, 2, 4\}$, $A_5 = \{1, 5\}$, $A_6 = \{1, 2, 3, 6\}$, $A_7 = \{1, 7\}$, and $A_{100} = \{1, 2, 4, 5, 10, 20, 25, 50, 100\}$.
c. $\{1\}$. **e.** $\{1\}$.

13. a. $A \cap B - C$.
c. $B \oplus C$.

14. $|A| + |B| + |C| + |D| - |A \cap B| - |A \cap C| - |A \cap D| - |B \cap C| - |B \cap D| - |C \cap D| + |A \cap B \cap C| + |A \cap B \cap D| + |A \cap C \cap D| + |B \cap C \cap D| - |A \cap B \cap C \cap D|$.

16. a. 82.
c. 23.

17. At most 20 drivers were smoking, talking, and tuning the radio.

19. a. $[x, y, z]$, $[x, y]$. **c.** $[a, a, a, b, b, c]$, $[a, a, b]$.
e. $[x, x, a, a, [a, a], [a, a]]$, $[x, x]$.

21. Let A and B be bags, and let m and n be the number of times x occurs in A and B, respectively. If $m \geq n$, then put $m - n$ occurrences of x in $A - B$, and if $m < n$, then do not put any occurrences of x in $A - B$.

22. a. $x \in A \cup \varnothing$ iff $x \in A$ or $x \in \varnothing$ iff $x \in A$. Therefore, $A \cup \varnothing = A$.
c. $x \in A \cup A$ iff $x \in A$ or $x \in A$ iff $x \in A$. Therefore, $A \cup A = A$.

23. a. Since there are no elements in $\varnothing$, there can be no elements in both A and $\varnothing$. Therefore, $A \cap \varnothing = \varnothing$.
c. $x \in A \cap (B \cap C)$ iff $x \in A$ and $x \in B \cap C$ iff $x \in A$ and $x \in B$ and $x \in C$ iff $x \in A \cap B$ and $x \in C$ iff $x \in (A \cap B) \cap C$. Therefore, $A \cap (B \cap C) = (A \cap B) \cap C$. **e.** First prove that $A \subset B$ implies $A \cap B = A$. Assume that $A \subset B$. If $x \in A \cap B$, then $x \in A$ and $x \in B$, and it follows that $x \in A$. Thus $A \cap B \subset A$. If $x \in A$, then $x \in B$ (by assumption), and it follows that $x \in A \cap B$. So $A \subset A \cap B$. Therefore, $A \cap B = A$. Next prove that $A \cap B = A$ implies $A \subset B$. Assume that $A \cap B = A$. Let $x \in A$. Then $x \in A \cap B$, which says $x \in A$ and $x \in B$. So $x \in B$. Therefore, we have $A \subset B$. So the iff statement has been proven.

24. Let $S \in$ power$(A \cap B)$. Then $S \subset A \cap B$, which says that $S \subset A$ and $S \subset B$. Therefore, $S \in$ power(A) and $S \in$ power(B), which says that $S \in$ power$(A) \cap$ power(B). This proves that power$(A \cap B) \subset$ power$(A) \cap$ power(B). The other containment is similar.

26. a. Let $x \in A \cap (B \cup A)$. Then $x \in A$, so we have $A \cap (B \cup \text{A}) \subset A$. For the other containment, let $x \in A$. Then $x \in B \cup A$. Therefore, $x \in A \cap (B \cup A)$, which says that $A \subset A \cap (B \cup A)$. This proves the equality by set containment. We can also prove the equality by using a property of intersection (1.6e) applied to the two sets A and $B \cup A$. Thus (1.6e) becomes $A \subset B \cup A$ if and only if $A \cap (B \cup A) = A$. Since we know that $A \subset B \cup A$ is always true, the equality follows.

27. Assume that $(A \cap B) \cup C = A \cap (B \cup C)$. If $x \in C$, then $x \in (A \cap B) \cup C = A \cap (B \cup C)$, which says that $x \in A$. Thus $C \subset A$. Assume that $C \subset A$. If $x \in (A \cap B) \cup C$, then $x \in A \cap B$ or $x \in C$. In either case it follows that $x \in A \cap (B \cup C)$ because $C \subset A$. Thus $(A \cap B) \cup C \subset A \cap (B \cup C)$. The other containment is similar. Therefore, $(A \cap B) \cup C = A \cap (B \cup C)$.

28. a. Counterexample: $A = \{a\}$, $B = \{b\}$.
c. Counterexample: $A = \{a\}$, $B = \{b\}$, $C = \{b\}$.

29. a. $x \in (A')'$ iff $x \in U$ and $x \notin A'$ iff $x \notin U - A$ iff $x \in A$.
c. $x \in A \cap A'$ means that $x \in A$ and $x \in U - A$. This says that $x \in A$ and $x \notin A$, which can't happen. Therefore, $A \cap A' = \varnothing$. To see that $A \cup A' = U$, observe that any element of U must be either in A or not in A. **e.** $x \in (A \cap B)'$ iff $x \notin A \cap B$ iff $x \notin A$ or $x \notin B$ iff $x \in A'$ or $x \in B'$ iff $x \in A' \cup B'$. Therefore, $(A \cap B)' = A' \cup B'$. **g.** $A \cup (A' \cap B) = (A \cup A') \cap (A \cup B) = U \cap (A \cup B) = A \cup B$.

Section 1.3

1. (x, x, x), (x, x, y), (x, y, x), (y, x, x), (x, y, y), (y, x, y), (y, y, x), (y, y, y).

2. a. $\{(a, a), (a, b), (b, a), (b, b), (c, a), (c, b)\}$. **c.** $\{(\,)\}$.
e. $\{(a, a), (a, b), (a, c), (b, a), (b, b), (b, c), (c, a), (c, b), (c, c)\}$.

3. $\langle\ \rangle$, $\langle a\rangle$, $\langle b\rangle$, $\langle a, a\rangle$, $\langle a, b\rangle$, $\langle b, a\rangle$, $\langle b, b\rangle$.

4. a. head($\langle a\rangle$) $= a$ and tail($\langle a\rangle$) $= \langle\ \rangle$.
c. head($\langle\langle a, b\rangle, c\rangle$) $= \langle a, b\rangle$ and tail($\langle\langle a, b\rangle, c\rangle$) $= \langle c\rangle$.

5. a. $\langle 24, 60, \langle 2, 3, 4, 6, 12\rangle\rangle$.
c. $\langle 14, 15, \langle\ \rangle\rangle$.

8. a. $LM = \{bba, ab, a, abbbba, abbab, abba, bbba, bab, ba\}$.
c. $L^0 = \{\Lambda\}$. **e.** $L^2 = LL = \{\Lambda, abb, b, abbabb, abbb, babb, bb\}$.

9. a. $L = \{b, ba\}$.
c. $L = \{a, b\}$.
e. $\{\Lambda, ba\}$.

10. a. $x = uvw$, where u, w $\in L$ and $v \in M$.
c. $x = \Lambda$ or $x = u_1 \ldots u_n$, where $u_k \in L \cup M$. **e.** $x = \Lambda$ or $x \in L$ or $x \in M$ or $x = (s_1 \ldots s_n)(u_1 \ldots u_m)$, where $s_k \in L$ and $u_k \in M$.

11. a. $\{a, b\}^* \cap \{b, c\}^* = \{b\}^*$
c. $\{a, b, c\}^* - \{a\}^*$ is the set of strings over $\{a, b, c\}$ that contain at least one b or at least one c.

12. a. $\{(1, 12), (2, 12), (3, 12), (4, 12), (6, 12), (12, 12)\}$.
c. $\{(2, 1, 1), (3, 1, 2), (3, 2, 1)\}$.
e. $U = \{(a, 1), (a, 2), (b, 1), (b, 2)\}$.

13. a. $\{z \mid$ (Michigan, $y, z) \in$ Borders for some $y\}$.
c. $\{x \mid (x,$ None, None$) \in$ Borders for some $x\}$.
e. $\{(x, z) \mid (x,$ Canada, $z) \in$ Borders$\}$.

14. a. $2(5^3) - 2(4^3) = 122$. **c.** $1(2)(4^4) = 512$.

15. Let U be the set of n-tuples over A. In other words, $U = A^n$. Let S be the subset of U whose n-tuples do not contain any occurrences of letters from the set B. So $S = (A - B)^n$. Then $U - S$ is the set of n-tuples over A that contain at least one occurrence of an element from B. We have $|U - S| = |U| - |S| = |A^n| - |(A - B)^n| = |A|^n - |A - B|^n = |A|^n - (|A| - |B|)^n$.

16. a. $(x, y) \in (A \cup B) \times C$ iff $x \in A \cup B$ and $y \in C$ iff ($x \in A$ or $x \in B$) and $y \in C$ iff $(x, y) \in A \times C$ or $(x, y) \in B \times C$ iff $(x, y) \in (A \times C) \cup (B \times C)$. Therefore, $(A \cup B) \times C = (A \times C) \cup (B \times C)$.
c. Prove that $(A \cap B) \times C = (A \times C) \cap (B \times C)$. $(x, y) \in (A \cap B) \times C$ iff $x \in A \cap B$ and $y \in C$ iff ($x \in A$ and $x \in B$) and $y \in C$ iff $(x, y) \in (A \times C) \cap (B \times C)$, which proves the statement.

17. a. The statement is true because $s\Lambda = \Lambda s = s$ for any string s.
c. $x \in L(M \cup N)$
  iff $x = yz$, where $y \in L$ and $z \in M \cup N$
  iff $x = yz$, where $y \in L$ and ($z \in M$ or $z \in N$)
  iff $x = yz$, where $yz \in LM$ or $yz \in LN$
  iff $x \in LM$ or $x \in LN$
  iff $x \in LM \cup LN$.
Therefore, $L(M \cup N) = LM \cup LN$. The second equality is proved the same way.

18. a. The statement is true because $A^0 = \{\Lambda\}$ for any language A.
c. First we'll prove $L^* = L^*L^*$. Since $L^* = L^*\{\Lambda\}$, we have $L^* = L^*\{\Lambda\} \subset L^*L^*$. So $L^* \subset L^*L^*$. If $x \in L^*L^*$, then $x = yz$, where $y, z \in L^*$. So $y \in L^m$ and $z \in L^n$ for some numbers m and n. Therefore, $x = yz \in L^{m+n} \subset L^*$. Thus $L^*L^* \subset L^*$. So we have the equality $L^* = L^*L^*$.

Next we'll prove the equality $L^* = (L^*)^*$. L^* is a subset of $(L^*)^*$ by definition. If $x \in (L^*)^*$, then there is a number n such that $x \in (L^*)^n$. So x is a concatenation of n strings, each one from L^*. So x is a concatenation of n strings, each from some power of L. Therefore, $x \in L^*$. Thus $(L^*)^* \subset L^*$. So we have the equality $L^* = (L^*)^*$.

19. a. Notice that $(3, 7) = \{\{3\}, \{3, 7\}\}$ and $(7, 3) = \{\{7\}, \{7, 3\}\}$ and that the two sets cannot be equal. **c.** $(\{a\}, b) = \{\{a\}, \{b\}\} = \{\{b\}, \{a\}\} = (\{b\}, a)$.

20. a. From Exercise 19 we have $(x, y) = S = \{\{x\}, \{x, y\}\}$. Therefore,

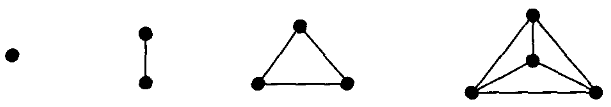
$$(x, y, z) = \{\{S\}, \{S, z\}\} = \{\{\{\{x\}, \{x, y\}\}\}, \{\{\{x\}, \{x, y\}\}, z\}\}.$$

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

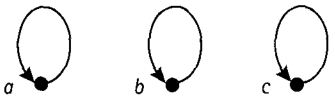
21. a. If a is a 3 by 4 matrix, then the address polynomial for the column-major location of $a[i, j]$ is $B + 3M(j - 1) + M(i - 1)$.
c. If a is a 3-dimensional array of size l by m by n stored as an l-tuple of m by n matrices, each in row-major form, then the address polynomial for the location of $a[i, j, k]$ is $B + mnM(i - 1) + nM(j - 1) + M(k - 1)$.

Section 1.4

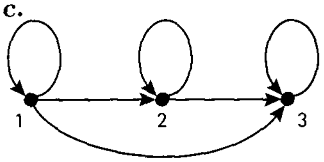
2.



4. a.



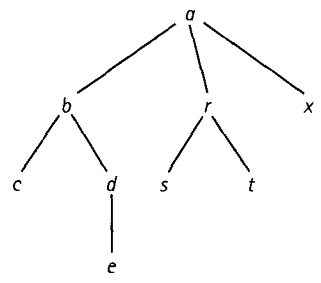
c.



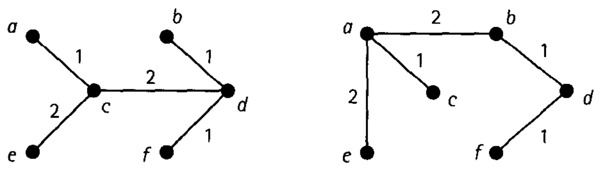
5. a. The four possible strings are $f\ d\ g\ e\ b\ a\ c$, $f\ d\ g\ b\ e\ a\ c$, $f\ g\ d\ e\ b\ a\ c$, and $f\ g\ d\ b\ e\ a\ c$.
b. The three possible strings are $f\ g\ e\ d\ b\ a\ c$, $f\ d\ e\ g\ b\ a\ c$, and $f\ d\ b\ a\ c\ e\ g$.

7. a. One answer is $a\ b\ c\ d\ e\ f$.
b. One answer is $a\ b\ c\ e\ d\ f$.

9.



11. Here are two answers:



13. The graph is connected and either none or two vertices have odd degree.

Chapter 2

Section 2.1

1. a. There is one function of type $\{a, b\} \to \{1\}$; it maps both a and b to 1.
c. There are four functions of type $\{a, b\} \to \{1, 2\}$: one maps both a and b to 1; one maps both a and b to 2; one maps a to 1 and b to 2; and one maps a to 2 and b to 1.

2. a. O. **c.** $\{x \mid x = 4k + 3 \text{ where } k \in \mathbb{N}\}$.
e. $\varnothing$.

3. a. –5.
c. 4.

4. a. 3.
c. 1.

5. $\gcd(296, 872) = 8 = (-53) \cdot 296 + 18 \cdot 872$.

6. a. 3.
c. 3.

7. a. $f(\varnothing) = \varnothing$.
c. $f(\{2, 5\}) = \{4\}$.
e. $f(\{1, 2, 3\}) = \{0, 2, 4\}$.

8. a. floor(x) = if $x \geq 0$ then trunc(x) else if x = trunc(x) then x else trunc$(x - 1)$.

9. When x is negative, $f(x, y)$ can be different than x mod y. For example, $f(-16, 3) = -1$ and -16 mod $3 = 2$.

11. a. 4.
c. -3.

12. a. If $x \in A \cup B$ then $x \in A$ or $x \in B$. So $\chi_{A\cup B}(x) = 1$ and either $\chi_A(x) = 1$ or $\chi_B(x) = 1$. If $\chi_A(x) = \chi_B(x) = 1$, the equation becomes $1 = 1 + 1 - 1(1)$, which is true. If $\chi_A(x) = 1$ and $\chi_B(x) = 0$, the equation becomes $1 = 1 + 0 - 1(0)$, which is true. If $x \notin A \cup B$, then $x \notin A$ and $x \notin B$. So $\chi_{A\cup B}(x) = \chi_A(x) = \chi_B(x) = 0$ and the equation becomes $0 = 0 + 0 - 0(0)$, which is true.
c. $\chi_{A-B}(x) = \chi_A(x)(1 - \chi_B(x))$.

13. a. A. **c.** $\{0\}$.

14. a. If x is an integer, then $\lfloor x\rfloor = x$ and $\lfloor x + 1\rfloor = x + 1$, which makes the desired equality true. If x is not an integer, then there is an integer n such that $n < x < n + 1$. Therefore, $\lfloor x\rfloor = n$ and $\lfloor x + 1\rfloor = n + 1$, which makes the desired equality true.
c. If $x \in \mathbb{Z}$, then certainly $\lceil x\rceil = \lfloor x\rfloor$. If $\lceil x\rceil = \lfloor x\rfloor$, then $\lfloor x\rfloor \le x \le \lceil x\rceil = \lfloor x\rfloor$, which says that $x = \lceil x\rceil = \lfloor x\rfloor$. Therefore, $x \in \mathbb{Z}$.

15. a. $\log_b 1 = 0$ means $b^0 = 1$, which is true.
c. $\log_b (b^x) = x$ means $b^x = b^x$, which is true.
e. Let $r = \log_b(x^y)$ and $s = \log_b x$, and proceed as in part (d) to show that $r = ys$. **g.** Let $r = \log_a x$, $s = \log_a b$, and $t = \log_b x$. Proceed as in (d) to show that $r = st$.

16. a. The equalities follow because gcd(a, b) is the largest common divisor of a and b.
c. Since gcd(d, a) = 1, we can use (2.1c) to find integers m and n such that $1 = md + na$. Multiply the equation by b to obtain $b = bmd + bna = bmd + abn$. Since d divides both terms on the right side, d also divides the left side. Therefore, $d \mid b$.

18. a. We'll prove both containments with iff statements: $x \in f(E \cup F)$ iff $x = f(y)$, where $y \in E \cup F$ iff $x = f(y)$, where $y \in E$ or $y \in F$ iff $x \in f(E)$ or $x \in f(F)$ iff $x \in f(E) \cup f(F)$.
c. Consider the function $f : \{a, b, c\} \to \{1, 2, 3\}$, defined by $f(a) = f(b) = 1$ and $f(c) = 2$. Then $\{a\} \cap \{b, c\} = \varnothing$, which gives $f(\{a\} \cap \{b, c\}) = f(\varnothing) = \varnothing$. But we have $f(\{a\}) \cap f(\{b, c\}) = \{1\}$. So $f(\{a\} \cap \{b, c\}) \ne f(\{a\}) \cap f(\{b, c\})$.

19. a. We'll prove both containments at once: $x \in f^{-1}(G \cup H)$ iff $f(x) \in G \cup H$ iff $f(x) \in G$ or $f(x) \in H$ iff $x \in f^{-1}(G)$ or $x \in f^{-1}(H)$ iff $x \in f^{-1}(G) \cup f^{-1}(H)$.
c. If $x \in E$, then $f(x) \in f(E)$, which says that $x \in f^{-1}(f(E))$. This proves the containment.
e. Consider the function $f : \{a, b, c\} \to \{1, 2, 3\}$, defined by $f(a) = f(b) = 1$ and $f(c) = 2$. Let $E = \{a\}$. Then $f^{-1}(f(E)) = f^{-1}(f(\{a\})) = f^{-1}(\{1\}) = \{a, b\}$. So E is a proper subset of $f^{-1}(f(E))$. For the other example, let $G = \{2, 3\}$. Then $f(f^{-1}(G)) = f(\{c\}) = \{2\}$. So $f(f^{-1}(G))$ is a proper subset of G.

20. a. By (2.4a) it suffices to show that n divides $(x + y) - ((x \text{ mod } n) + (y \text{ mod } n))$. By the definition of mod we have $x \text{ mod } n = x - nq_1$ and $y \text{ mod } n = y - nq_2$. So $(x + y) - ((x \text{ mod } n) + (y \text{ mod } n)) = (x + y) - ((x - nq_1) + (x - nq_2)) = n(q_1 - q_2)$. So n divides $(x + y) - ((x \text{ mod } n) + (y \text{ mod } n))$. The result follows from part (a).
c. Since $\text{gcd}(a, n) = 1$, it follows from (2.2c) that there are integers x and y such that $1 = ax + ny$. Now we have the sequence of equations

$$\begin{aligned}
1 \text{ mod } n &= (ax + ny) \text{ mod } n \\
&= ((ax \text{ mod } n) + (ny \text{ mod } n)) \text{ mod } n && \text{(by (a))} \\
&= ((ax \text{ mod } n) + ((n \text{ mod } n)(y \text{ mod } n) \text{ mod } n)) \text{ mod } n && \text{(by (b))} \\
&= ((ax \text{ mod } n) + 0) \text{ mod } n \\
&= ax \text{ mod } n.
\end{aligned}$$

21. Let $a = dq + r$, where $0 \leq r < d$. Solve for r to obtain $r = a - dq$. Now use the fact that $d = ax + by$ to substitute for d to obtain

$$\begin{aligned}
r &= a - dq \\
&= a - (ax + by)q \\
&= a(1 - xq) + b(-yq)
\end{aligned}$$

which, if $r > 0$, has the form of a number in S. But d is the smallest number in S and $0 \leq r < d$. So if $r > 0$, then it would be a number in S smaller than the smallest number, a contradiction. Therefore, $r = 0$ and consequently $d|a$. Similarly, we have $d|b$. QED.

Section 2.2

1. a. 4. **c.** 2. **e.** $\langle(4, 0), (4, 1), (4, 2), (4, 3)\rangle$. **g.** $\langle(+, (0, 0)), (+, (1, 1)), (+, (2, 2))\rangle$.

2. a. $f(g(x)) = \text{ceiling}(x)$, $g(f(x)) = (2)\text{ceiling}(x/2)$, $f(g(1)) = 1$, and $g(f(1)) = 0$.
c. $f(g(x)) = \text{gcd}(x \text{ mod } 5, 10)$, $g(f(x)) = \text{gcd}(x, 10) \text{ mod } 5$, $f(g(5)) = 10$, and $g(f(5)) = 5$.

3. a. $f(g(x, y))$. **c.** $f(g(x, g(y, z)))$.

4. a. $2^7 \leq x < 2^8$.

5. One solution is $\text{max4}(w, x, y, z) = \text{max}(\text{max}(\text{max}(w, x), y), z)$.

6. $\text{floor}(\log_2(x)) + 1$.

7. a. $\langle 0, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4\rangle$.

8. a. $f(n) = \text{map}(+, \text{pairs}(\text{seq}(n), \text{seq}(n)))$.

9. a. $f(n, k) = \text{map}(+, \text{dist}(n, \text{seq}(k)))$.
c. $f(n, m) = \text{map}(+, \text{dist}(n, \text{seq}(m - n)))$
e. $f(n) = \text{pairs}(\text{seq}(n), g(n))$, where $g(n)$ is the solution to part (d).

g. $f(g, n) = \text{pairs}(\text{seq}(n), \text{map}(g, \text{seq}(n)))$.
i. $f(g, h, \langle x_1, \ldots, x_n\rangle) = \text{pairs}(\text{map}(g, \langle x_1, \ldots, x_n\rangle), \text{map}(h, \langle x_1, \ldots, x_n\rangle))$.

10. $f(n) = \text{map}(+, \text{dist}(1, \text{seq}(n - 1)))$.

11. a. The ceiling returns an integer, and the floor of an integer is itself.
c. Any x is in $2^n \leq x < 2^{n+1}$ for some integer n. Therefore, $2^n \leq \text{floor}(x) < 2^{n+1}$. Take the log to get $n \leq \log_2(x) < n + 1$ and $n \leq \log_2(\text{floor}(x)) < n + 1$. So $\text{floor}(\log_2(x)) = n = \text{floor}(\log_2(\text{floor}(x)))$.

Section 2.3

1. The fatherOf function is not injective because some fathers have more than one child. For example, if John and Mary have the same father, then fatherOf(John) = fatherOf(Mary). The fatherOf function is not surjective because there are people who are not fathers. For example, Mary is not a father. So fatherOf(x) $\neq$ Mary for all people x
2. a. $f : C \to B$, where $f(1) = x$, $f(2) = y$.
c. $f : A \to B$, where $f(a) = x$, $f(b) = y$, $f(c) = z$.

3. a. Eight functions; no injections, six surjections, no bijections, and two with none of the properties.
c. 27 functions; six satisfy the three properties (injective, surjective, and bijective), 21 with none of the properties.

4. a. If $f(x) = f(y)$, then $2x = 2y$, which upon dividing by 2 yields $x = y$. So f is injective. f is not surjective because the range of f is the set of even natural numbers, which is not equal to the codomain $\mathbb{N}$. e.g., no x maps to 1.
c. If $y \in \mathbb{N}$, then $f(2y) = \text{floor}(2y/2) = \text{floor}(y) = y$. So f is surjective. f is not injective because, for example, $f(0) = f(1) = 0$.
e. Let $f(x) = f(y)$. If x is odd and y is even, then $x - 1 = y + 1$, which implies $x = y + 2$. This tells us that x and y are either both even or both odd, a contradiction. We get a similar contradiction if x is even and y is odd. If x and y are both even, then $x + 1 = y + 1$, which impies $x = y$. If x and y are both odd, then $x - 1 = y - 1$, which implies that $x = y$. So f is injective. Let $y \in \mathbb{N}$. If y is odd, then $y - 1$ is even and $f(y - 1) = y - 1 + 1 = y$. If y is even, then $y + 1$ is odd and $f(y + 1) = y + 1 - 1 = y$. So f is surjective. Therefore, f is bijective.

5. a. Surjective. **c.** Surjective. **e.** Injective. **g.** Surjective. **i.** Surjective.

6. a. Let $f(x) = f(y)$. Then $(b - a)x + a = (b - a)y + a$. Subtract a from both sides and divide the resulting equation by $(b - a)$ to obtain $x = y$. So f is injective. To show f is surjective, let $y \in (a, b)$ and solve the equation $f(x) = y$ for x to obtain $(b - a)x + a = y$, which gives $x = (y - a)/(b - a)$. It follows that $f((y - a)/(b - a)) = y$ and since $a < y < b$, we have $0 < (y - a)/(b - a) < 1$. Thus f is surjective. Therefore, f is a bijection.
c. Let $f(x) = f(y)$. Then $1/(2x - 1) - 1 = 1/(2y - 1) - 1$, which by elementary algebra implies that $x = y$. Therefore, f is an injection. To show f is surjective, let $y > 0$ and then find some x such that $f(x) = y$. Solve the equation $f(x) = y$ to get $x = (2 + y)/(2y + 2)$. It follows that $f((2 + y)/(2y + 2)) = y$, and since

$y > 0$, it follows that $1/2 < (2 + y)/(2y + 2) < 1$. To see this, we can obtain a contradiction if $(2 + y)/(2y + 2) < 1/2$ or if $(2 + y)/(2y + 2) > 1$. So f is surjective. Therefore, f is a bijection.
e. f is a bijection because it is defined in terms of the bijections given in parts (c) and (d). To see this, let $f(x) = f(y)$. If this value is positive, then $x = y$ by part (c). If the value is negative, then $x = y$ by part (d). If $f(x) = f(y) = 0$, then $x = y = 1/2$ by the definition of f. So f is injective. To show f is surjective, let $y \in \mathbb{R}$ and find some $x \in (0, 1)$ such that $f(x) = y$. Again, take the three cases. If $y > 0$, then part (c) gives an element $x \in (1/2, 1)$ such that $f(x) = y$. If $y < 0$, then part (d) gives an element $x \in (0, 1/2)$ such that $f(x) = y$. If $y = 0$, then $f(1/2) = y$. So f is surjective. Therefore, f is a bijection.
7. a. 15.
c. Any nonempty string over $\{a, b, c\}$ has one of nine possible patterns of beginning, ending letters. So any set of 10 such strings will contain two strings with the same beginning ending pairs.

8. a. Since there are 10 decimal digits, we can be assured that any set of 11 numbers will contain two numbers that use a common digit in their representations.
c. Of the ten numbers listed, at least nine are in the range from 1 to 8. (It could happen that some $x_k = 8$, so that $x_k + 1 = 9$ is not in the set.) So the pigeonhole principle tells us that two of the nine numbers are equal. Since the two lists are each distinct, it follows that $x_i = x_j + 1$ for some i and j.

9. a. Not bijective because $\gcd(2, 6) \neq 1$. The fixed point is 0.
c. Bijective and $f^{-1} = f$. The fixed points are 0 and 3.
e. Bijective and $f^{-1}(x) = (4x + 2) \bmod 7$. The fixed point is 4.
g. Bijective and $f^{-1}(x) = (9x + 1) \bmod 16$. There are no fixed points.

10. a. $\{a \mid \gcd(a, 26) = 1\} = \{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25\}$.

11. a. one, two, six, four, five, nine, three, seven, eight.
c. Only one, two, and three can be placed in the table: one, blank, blank, two, blank, blank, three, blank, blank.

12. a. Wednesday, Monday, Friday, Tuesday, Sunday, Thursday, Saturday.
c. Wednesday, Monday, Sunday, Tuesday, Friday, Thursday, Saturday.

13. a. March, April, January, February, July, August, May, June.
c. March, May, January, February, August, July, April, June.

15. a. Let f and g be injective, and assume that $g \circ f(x) = g \circ f(y)$ for some $y \in A$. Since g is injective, it follows that $f(x) = f(y)$, and it follows that $x = y$ because f is injective. Therefore, $g \circ f$ is injective.
c. If f and g are bijective, then they are both injective and surjective. So by parts (a) and (b) the composition $g \circ f$ is injective and surjective, hence bijective.

16. Let $x \in A$ and let $f(g(x)) = y$. Apply g to both sides to obtain $g(f(g(x))$ $= g(y)$. Since $g(f(x)) = x$ for all x, it follows that $g(f(g(x))) = g(x)$. So $g(x)$ $= g(y)$. Since g is bijective, hence injective, it follows that $x = y$. Therefore,

$f(g(x)) = x$.

17. a. If $g \circ f$ is surjective, then for each element $z \in C$ there exists an element $x \in A$ such that $z = (g \circ f)(x) = g(f(x))$. So it follows that $f(x)$ is an element of B such that $z = g(f(x))$. Therefore, g is surjective if $g \circ f$ is surjective.

18. a. Let f be surjective, and let $b \in B$ and $c \in C$. Then there exists an element $a \in A$ such that $f(a) = (b, c)$. But $f(a) = (g(a), h(a))$. Therefore, $b = g(a)$ and $c = h(a)$. So g and h are surjective. Now let $A = \{1, 2, 3\}$, $B = \{4, 5\}$, and $C = \{6, 7\}$. The set $B \times C$ has four elements, and A has three elements. So there can be no surjection from A to $B \times C$.

19. Let $g = \gcd(a, n)$. Let x be an integer such that $ax \bmod n = b \bmod n$. Then n divides $(ax - b)$. So there is an integer q such that $ax - b = nq$, or $b = a - nq$. Since g divides a and g divides n, it follows from (1.1b) that g divides b. For the converse, suppose that g divides b. Then we can write $b = gk$ for some integer k. Since $g = \gcd(a, n)$ it follows from (2.2c) that $g = as + nt$ for some integers s and t. Multiply this equation by k to obtain $b = gk = ask + ntk$. Apply mod n to both sides to obtain $b \bmod n = ask \bmod n$. Therefore, $x = sk$ is a solution to the equation $ax \bmod n = b \bmod n$.

21. If we show that g is a bijection and $f(g(x)) = g(f(x)) = x$ for all $x \in \mathbb{N}_n$, then it follows that $g = f^{-1}$. Since $1 = ak + nm$, it follows that $\gcd(k, n) = 1$. So g is a bijection by the first part of (2.6). We have the following sequence of equations.

$$\begin{aligned}
f(g(x)) &= (ag(x) + b) \bmod n \\
&= (a((kx + c) \bmod n) + b) \bmod n \\
&= (a(kx + c + nq) + b) \bmod n && \text{(for some integer } q\text{)} \\
&= (akx + ac + b) \bmod n && \text{(by (2.4))} \\
&= (akx + f(c)) \bmod n \\
&= (akx + 0) \bmod n && (f(c) = 0) \\
&= (akx) \bmod n \\
&= ((1 - nm)x) \bmod n && (1 = ak + nm) \\
&= (x - nmx) \bmod n \\
&= x \bmod n \\
&= x.
\end{aligned}$$

Note also that if we let $g(f(x)) = y$, then we can apply f to both sides to get $f(g(f(x))) = f(y)$. But $f(g(f(x))) = f(x)$ because we just showed that $f(g(x)) = x$ for all $x \in \mathbb{N}_n$. So $f(x) = f(y)$, from which we conclude that $x = y$ because f is injective. Therefore, $g(f(x)) = x$ for all $x \in \mathbb{N}_n$. So $g = f^{-1}$. QED.

Section 2.4

1. a. Let A be the set. The smallest number in A is $2(0) + 5 = 5$ and the largest number in A is $2(46) + 5 = 97$. So the function $f : \{0, 1, \ldots, 46\} \to A$ defined by mapping $f(x) = 2x + 5$ is a bijection. Therefore, $|A| = 47$.

c. The function $f : \{0, 1, \ldots, 15\} \to \{2, 5, 8, 11, 14, 17, \ldots, 44, 47\}$ defined by $f(k) = 2 + 3k$ is a bijection. So the cardinality of the set is 16.

2. a. Let Even be the set of even natural numbers. For example, the function $f : \mathbb{N} \to$ Even defined by $f(k) = 2k$ is a bijection. So Even is countable.
c. Let S be the set of strings over $\{a\}$. So $S = \{a\}^* = \{\Lambda, a, aa, aaa, \ldots\}$. The mapping from $\mathbb{N}$ to S that maps each n to the string of length n is a bijection. So S is countable.
e. For example, the function $f : \mathbb{Z} \to \mathbb{N}$ defined $f(x) = 2x - 1$ when $x > 0$ and $f(x) = -2x$ when $x \leq 0$ is a bijection. So $\mathbb{Z}$ is countable.
g. Let E be the set of even integers. For example, the function $f : \mathbb{N} \to E$ defined $f(x) = x - 1$ when x is odd and $f(x) = -(x + 2)$ when x is even is a bijection. So E is countable.

3. a. Let S be the set of strings over $\{a, b\}$ that have odd length. For each number n let S_n be the set of all strings over $\{a, b\}$ that have length $2n + 1$. For example, $S_0 = \{a, b\}$ and $S_1 = \{aaa, aab, aba, baa, bbb, bba, bab, abb\}$. It follows that $S = S_0 \cup S_1 \cup \ldots \cup S_n \cup \ldots$. Since each set S_n is finite, hence countable, it follows from (2.10) that the union is countable.
c. Let B be the set of all binary trees over $\{a, b\}$. For each natural number n let S_n be the set of all binary trees over $\{a, b\}$ that have n nodes. It follows that $B = S_0 \cup S_1 \cup \ldots \cup S_n \cup \ldots$. Since each set S_n is finite, hence countable, it follows from (2.10) that the union is countable.

4. a. Let $g(n) =$ hello if $f_n(n) =$ world, and let $g(n) =$ world if $f_n(n) =$ hello. Then the sequence $(g(0), g(1), \ldots, g(n), \ldots)$ is not in the given set.
c. Let $g(n) = 2$ if $a_{nn} = 4$, and let $g(n) = 6$ if $a_{nn} \neq 4$. Then the sequence $(g(0), g(1), \ldots, g(n), \ldots)$ is not in the given set.

5. We can represent each subset S of $\mathbb{N}$ as a sequence of 1's and 0's where 1 in the kth position means that $k \in S$ and 0 means $k \notin S$. For example, $\mathbb{N}$ is represented by $(1, 1, 1, \ldots)$ and the empty set by $(0, 0, 0, \ldots)$. So each set S_n can be represented by an infinite sequence of 0's and 1's. But now (2.12) applies to say that there is some sequence of 1's and 0's that is not listed. This contradicts the statement that all subsets of $\mathbb{N}$ are listed.

7. a. Let S be a subset of the countable set A. Then the mapping from S to A that sends every element to itself is an injection. So $|S| \leq |A|$. Since A is countable, there is an injection from A to $\mathbb{N}$ by countable property (b). Since a composition of injections is an injection, we have an injection from S to $\mathbb{N}$. Therefore, $|S| \leq |\mathbb{N}|$.

8. a. For each natural number k, let S_k be the set of strings of length n over the alphabet $\{a_0, a_1, \ldots, a_k\}$. It follows that $A_n = S_0 \cup S_1 \cup \ldots \cup S_n \cup \ldots$. Since each set S_n is finite, hence countable, it follows from (2.10) that the union is countable.

9. For each n let F_n be the collection of subsets of $\{0, 1, \ldots, n\}$. In other words, $F_n =$ power$(\{0, 1, \ldots, n\})$. Since any finite subset S of $\mathbb{N}$ has a largest element n, it follows that S is in the collection F_n. So Finite($\mathbb{N}$) $= F_0 \cup F_1 \cup \ldots \cup F_n \cup \ldots$. Each set F_n is countable because it is finite. So (2.10) tells us that the union is countable.

Chapter 3

Section 3.1

1. **a.** 3, 5, 9, 17, 33, 65, 129, 257, 513, 1025.

2. **a.** Basis: $1 \in S$; Induction: If $x \in S$, then $x + 2 \in S$.
c. Basis: $-3 \in S$; Induction: If $x \in S$, then $x + 2 \in S$.
e. Basis: $1 \in S$; Induction: If $x \in S$, then $(\sqrt{x} + 1)^2 \in S$.

3. **a.** Basis: $4, 3 \in S$. Induction: If $x \in S$, then $x + 3 \in S$.

4. **a.** Basis: $0, 1 \in S$; Induction: If $x \in S$, then $x + 4 \in S$.
c. Basis: $2 \in S$; Induction: If $x \in S$, then $x + 5 \in S$.

5. $4 = 3 \cup \{3\} = 2 \cup \{2\} \cup \{3\} = 1 \cup \{1\} \cup \{2\} \cup \{3\} = 0 \cup \{0\} \cup \{1\} \cup \{2\} \cup \{3\} = \varnothing \cup \{0\} \cup \{1\} \cup \{2\} \cup \{3\} = \{0, 1, 2, 3\}$.

6. **a.** Basis: $b \in S$. Induction: If $x \in S$, then $axc \in S$.
c. Basis: $a \in S$. Induction: If $x \in S$, then $aax \in S$.
e. Basis: $b \in S$. Induction: If $x \in S$, then ax, $xc \in S$.
g. Basis: $b \in S$. Induction: If $x \in S$, then ax, $xb \in S$.
i. Basis: a, $b \in S$. Induction: If $x \in S$, then ax, $xb \in S$.
k. Basis: $\Lambda \in S$. Induction: If $x \in S$, then abx, bax, axb, $bxa \in S$.

7. **a.** Basis: $\Lambda \in S$. Induction: If $x \in S$, then axa, $bxb \in S$.
c. Basis: Λ, a, $b \in S$. Induction: If $x \in S$, then axa, $bxb \in S$.

8. Basis: a, b, c, x, y, $z \in T$. Induction: If $t \in T$, then $f(t)$, $g(t) \in T$.

9. **a.** $\langle a \rangle$, $\langle b, a \rangle$, $\langle b, b, a \rangle$, $\langle b, b, b, a \rangle$, $\langle b, b, b, b, a \rangle$.

10. **a.** Basis: $\langle a \rangle \in S$. Induction: If $L \in S$, then $a :: L \in S$.
c. Basis: $\langle a, b \rangle$, $\langle b, a \rangle \in S$. Induction: If $L \in S$, then head(L) $:: L \in S$.
e. Basis: $\langle\ \rangle \in S$. Induction: If $L \in S$ and a, $b \in \{0, 1, 2\}$, then $a :: b :: L \in S$.
g. Basis: $\langle a \rangle \in S$. Induction: If $L \in S$, then $a :: a :: L \in S$.
i. Basis: $\langle a \rangle \in S$ for all $a \in A$. Induction: If $L \in S$ and a, $b \in A$, then $a :: b :: L \in S$.

11. **a.** Basis: $\langle a \rangle \in S$. Induction: If $L \in S$, then consR(L, b) $\in S$.
c. Basis: $\langle\ \rangle \in S$. Induction: If $L \in S$, then put the following four lists in S: cons(a, cons(b, L)), cons(b, cons(a, L)), cons(a, consR(L, b)), and cons(b, consR(L, a)).

13. Each nonleaf node has a leaf as the left child and a tree with the same property as the right child.

15. Basis: tree($\langle\ \rangle$, a, $\langle\ \rangle$) $\in B$. Induction: If $T \in B$, then tree(T, a, tree($\langle\ \rangle$, a, $\langle\ \rangle$)), tree(tree($\langle\ \rangle$, a, $\langle\ \rangle$), a, T) $\in B$.

16. **a.** $B = \{(x, y) \mid x, y \in \mathbb{N} \text{ and } x \geq y\}$.

17. a. Basis: $(0, 0) \in S$.
Induction: If $(x, y) \in S$ and $x = y$, then $(x, y + 1)$, $(x + 1, y + 1) \in S$.

18. a. Basis: $(\langle\ \rangle, \langle\ \rangle) \in S$. Induction: If $(x, y) \in S$ and $a \in A$, then $(a :: x, y)$, $(x, a :: y) \in S$.
c. Basis: $(0, \langle\ \rangle) \in S$. Induction: If $(x, L) \in S$ and $m \in \mathbb{N}$, then $(x, m :: L)$, $(x + 1, L) \in S$.

19. Basis: $\langle\ \rangle \in E$ and $\langle a \rangle \in O$ for all $a \in A$. Induction:
If S, $T \in E$ and $a \in A$, then tree$(S, a, T) \in O$.
If S, $T \in O$ and $a \in A$, then tree$(S, a, T) \in O$.
If $S \in E$ and $T \in O$ and $a \in A$, then tree(S, a, T), tree(T, a, S) in E.

20. Basis: $(a, g(a)) \in A$.
Induction: If $(x, y) \in A$ and $y < f(x)$, then $(x, y + 1) \in A$.
If $(x, y) \in A$ and $x < b$, then $(x + 1, g(x + 1)) \in A$.

Section 3.2

1. We'll evaluate the leftmost term in each expression.
fib(4) = fib(3) + fib(2) = fib(2) + fib(1) + fib(2) = fib(1) + fib(0) + fib(1) + fib(2) = 1 + fib(0) + fib(1) + fib(2) = 1 + 0 + fib(1) + fib(2) = 1 + 0 + 1 + fib(2) = 1 + 0 + 1 + fib(1) + fib(0) = 1 + 0 + 1 + 1 + fib(0) = 1 + 0 + 1 + 1 + 0 = 3.

3. For (3.9): $\text{makeTree}(\langle\ \rangle, \langle 3,2,4\rangle) = \text{makeTree}(\text{insert}(3, \langle\ \rangle), \langle 2,4\rangle)$
$= \text{makeTree}(\text{insert}(2, \text{insert}(3, \langle\ \rangle)), \langle 4\rangle)$
$= \text{makeTree}(\text{insert}(4, \text{insert}(2, \text{insert}(3, \langle\ \rangle))), \langle\ \rangle)$
$= \text{insert}(4, \text{insert}(2, \text{insert}(3, \langle\ \rangle)))$.
For (3.10): $\text{makeTree}(\langle\ \rangle, \langle 3,2,4\rangle = \text{insert}(3, \text{makeTree}(\langle\ \rangle, \langle 2,4\rangle))$
$= \text{insert}(3, \text{insert}(2, \text{makeTree}(\langle\ \rangle, \langle 4\rangle)))$
$= \text{insert}(3, \text{insert}(2, \text{insert}(4, \text{makeTree}(\langle\ \rangle, \langle\ \rangle))))$
$= \text{insert}(3, \text{insert}(2, \text{insert}(4, \langle\ \rangle)))$.

4. a. $f(0) = 0$ and $f(n) = f(n - 1) + 2n$.
c. $f(1, n) = gcd(1, n)$ and $f(k, n) = f(k - 1, n) + gcd(k, n)$.
e. $f(0, k) = 0$ and $f(n, k) = f(n - 1, k) + nk$.

5. a. $f(\Lambda) = \Lambda$ and $f(ax) = f(x)a$ and $f(bx) = f(x)b$.
c. $f(x, y) =$ if $x = \Lambda$ then true
else if $x = as$ and $y = at$ or $x = bs$ and $y = bt$ then $f(s, t)$
else false.
e. $f(x) =$ if $x = \Lambda$ or $x = a$ or $x = b$ then true
else if $x = asa$ or $x = bsb$ then $f(s)$
else false.

6. a. $f(0) = \langle 0\rangle$ and $f(n) = 2n :: f(n - 1)$.
c. $f(x, \langle\ \rangle) = 0$ and $f(x, h :: t) = h + xf(x, t)$.
e. $f(a, \langle\ \rangle) = \langle\ \rangle$ and $f(a, h :: t) = (h + a) :: f(a, t)$.
g. $f(0) = \langle(0, 0)\rangle$ and $f(n) = (0, n) :: g(1, f(n - 1))$, where g adds 1 to the first component of each ordered pair in a list of ordered pairs. (See part (f).)
i. $f(g, h, \langle\ \rangle) = \langle\ \rangle$ and $f(g, h, k :: t) = (g(k), h(k)) :: f(g, h, t)$.

7. **a.** $f(0) = \langle 0 \rangle$ and $f(n+1) = \text{cat}(f(n), \langle n+1 \rangle)$.
c. $f(0) = \langle 1 \rangle$ and $f(n) = \text{cat}(f(n-1), \langle 2n+1 \rangle)$.
e. $f(0,k) = \langle 0 \rangle$ and $f(n,k) = \text{cat}(f(n-1,k), \langle nk \rangle)$.
g. $f(n, n) = \langle n \rangle$ and $f(n, m) = \text{cat}(f(n, m-1), \langle m \rangle)$.

8. $\text{insert}(f, \langle a, b \rangle) = f(a, b)$,
$\text{insert}(f, \text{cons}(h, t)) = f(a, \text{insert}(f, t))$.

10. **a.** $\text{last}(\langle x \rangle) = x$,
$\text{last}(\text{cons}(h, t)) = \text{last}(t)$.

12. Let rem(L) denote the list obtained from L by removing repetitions of elements and keeping the rightmost occurrence of each element.

```
rem (L) = if L = ⟨ ⟩ then ⟨ ⟩
              else cat(rem(removeAll(last (L) , front (L) )), last (L) :: ⟨ ⟩ ).
```

14. **a.** In(T): **if** $T \neq \langle\rangle$ **then** In(left(T)); print(root(T)); In(right(T)) **fi.**

15. **a.** Equational form:

```
leaves(⟨ ⟩) = 0,
leaves(tree(⟨ ⟩, a, ⟨ ⟩)) = 1,
leaves(tree(l, a, r)) = leaves(l) + leaves(r).
```

If-then form:

```
leaves(t) = if t = ⟨ ⟩ then 0
            else if left(t) = right(t) = ⟨ ⟩ then 1
            else leaves(left(t)) + leaves(right(t)).
```

c. Equational form:

```
postOrd(⟨ ⟩) = ⟨ ⟩
postOrd(tree(L, r, R)) = cat(postOrd(L), cat(postOrd(R), ⟨ r⟩)).
```

If-then form:

```
postOrd(T) = if T = ⟨ ⟩ then ⟨ ⟩
             else cat(postOrd(left(T)), cat(postOrd(right(T)), ⟨root(T)⟩)).
```

16. **a.** $f(\langle\ \rangle) = \langle\ \rangle$ and $f(\langle L, r, R \rangle) = r + f(L) + \text{f}(R)$.
c. $f(\langle\ \rangle) = \langle\ \rangle$ and $f(\langle L, r, R \rangle\ ) =$ if $p(r)$ then $r :: \text{cat}(f(L), f(R))$ else $\text{cat}(f(L), f(R))$.

18. **a.** $f(k, \langle\ \rangle) = \langle\ \rangle$ and $f(k, x :: t) = x_k :: f(k, t)$.

19. **a.**

```
isMember(x, L) = if L = ⟨ ⟩ then false
                 else if x = head(L) then true
                 else isMember(x, tail(L)).
```

c. areEqual(K, L) = if isSubset(K, L) then isSubset(L, K) else false.

e.

```
intersect(K, L) = if K = ⟨ ⟩ then ⟨ ⟩
                  else if isMember(head(K), L) then
                     head(K) :: intersect(tail(K), L)
                  else intersect(tail(K), L).
```

20. 1, 1, 2, 2, 3, 4, 4, 4, 5, 6, 7, 7, 8, 8, 8, 8, 9.

22. Assume that the product of the empty list $\langle\ \rangle$ with any list is $\langle\ \rangle$. Then define product as follows:

$$\begin{aligned}\text{product}(A, B) = {} & \text{if } A = \langle\ \rangle \text{ or } B = \langle\ \rangle \text{ then } \langle\ \rangle\\ & \quad \text{else concatenate the four lists}\\ & \qquad \langle(\text{head}(A), \text{head}(B))\rangle,\\ & \qquad \text{product}(\langle\text{head}(A)\rangle, \text{tail}(B)),\\ & \qquad \text{product}(\text{tail}(A), \langle\text{head}(B)\rangle), \text{ and}\\ & \qquad \text{product}(\text{tail}(A), \text{tail}(B)).\end{aligned}$$

23. a. 1, 2.5, 2.05. **c.** 3, 2.166..., 2.0064... .
e. 1, 5, 3.4.

24. a. Square(x :: s) = x^2 :: Square(s).
c. Prod(n, s) = if $n = 0$ then 1 else head(s)*Prod($n - 1$, tail(s)).
e. Skip(x, k) = x :: Skip($x + k$, k).
g. ListOf(n, s) = if $n = 0$ then $\langle\ \rangle$ else head(s) :: ListOf($n - 1$, tail(s)).

25. a. head(Primes) = head(sieve(inst(2))) = head(sieve(2 :: ints(3))) = head(2 :: sieve(remove(2, ints(3)))) = 2
c. remove(2, ints(0))) = remove(2, 0 :: ints(1)) = remove(2, ints(1))
= remove(2, 1 :: ints(2)) = 1 :: remove(2, ints(2)) = 1 :: remove(2, 2 :: ints(3))
= 1 :: remove(2, ints(3)) = 1 :: remove(2, 3 :: ints(4))
= 1 :: 3 :: remove(2, ints(4)).

26. $f(x)$ = if $x \leq 10$ then 1 else $x - 10$.

Section 3.3

1. a. $S \to DS$, $D \to 7$, $S \to DS$, $S \to DS$, $D \to 8$, $D \to 0$, $S \to D$, $D \to 1$.
c. $S \Rightarrow DS \Rightarrow DDS \Rightarrow DDDS \Rightarrow DDDD \Rightarrow DDD1 \Rightarrow DD01 \Rightarrow D801 \Rightarrow 7801$.

2. a. Leftmost: $S \Rightarrow S[S] \Rightarrow [S] \Rightarrow [\,]$. Rightmost: $S \Rightarrow S[S] \Rightarrow S[\,] \Rightarrow [\,]$.
c. Leftmost: $S \Rightarrow S[S] \Rightarrow S[S]\,[S] \Rightarrow [S]\,[S] \Rightarrow [\,]\,[S] \Rightarrow [\,]\,[\,]$.
Rightmost: $S \Rightarrow S[S] \Rightarrow S[\,] \Rightarrow S[S]\,[\,] \Rightarrow S[\,]\,[\,] \Rightarrow [\,]\,[\,]$.

3. a. $S \to bb \mid bbS$.
c. $S \to \Lambda \mid abS$.
e. $S \to ab \mid abS$.
g. $S \to b \mid bbS$.
i. $S \to aBc$ and $B \to \Lambda \mid bB$.

4. a. $S \to AB$ and $A \to \Lambda \mid aA$ and $B \to \Lambda \mid bB$.
c. $S \to AB$ and $A \to a \mid aA$ and $B \to \Lambda \mid bB$.
e. $S \to AB$ and $A \to a \mid aA$ and $B \to b \mid bB$.

5. a. $S \to \Lambda \mid aSa \mid bSb \mid cSc$.
c. $S \to A \mid B$ and $A \to \Lambda \mid aaA$ and $B \to b \mid bbB$.
e. $S \to aAB \mid ABb$ and $A \to \Lambda \mid aA$ and $B \to \Lambda \mid bB$.

6. a. $O \to B1$ and $B \to \Lambda \mid B0 \mid B1$.

c.$O \to D1 \mid D3 \mid D5 \mid D7 \mid D9$ and $D \to \Lambda \mid D0 \mid D1 \mid D2 \mid D3 \mid D4 \mid D5 \mid D6 \mid D7 \mid D8 \mid D9$.

7. a. $S \to D \mid S + S \mid (S)$, and D denotes a decimal numeral.

8. a. $S \to a \mid b \mid c \mid x \mid y \mid z \mid f(S) \mid g(S)$.

9. a. $S \to a \mid b \mid c \mid x \mid y \mid z \mid f(S) \mid g(S, S)$.

11. a. The string *ababa* has two parse trees.
c. The string *aa* has two parse trees.
e. The string [ ] [ ] has two parse trees.

13. a. $S \to a \mid abS$.
c. $S \to a \mid aS$.
e. $S \to S[S] \mid \Lambda$.

14. a. $S \to A \mid AB$ and $A \to Aa \mid a$ and $B \to Bb \mid b$.

15. a. Basis: $\Lambda \in L(G)$. Induction: If $w \in L(G)$, then put $aaw \in L(G)$.

Chapter 4

Section 4.1

1. a. Reflexive, symmetric, transitive. **c.** Reflexive, symmetric, transitive. **e.** Reflexive, symmetric, transitive. **g.** Irreflexive, transitive. **i.** Reflexive.

2. a. Symmetric. **c.** Reflexive, antisymmetric, and transitive.
e. Symmetric.

3. a. The irreflexive follows because $(x, x) \notin \varnothing$ for any x. The symmetric, antisymmetric, and transitive properties are conditional statements that are always true because their hypotheses are false.
b. $A \times A$ is reflexive, symmetric, and transitive because it contains all possible pairs. If $A = \{a\}$, then $A \times A = \{(a, a)\}$, which is also antisymmetric.

4. a. $\{(a, a), (b, b), (c, c), (a, b), (b, c)\}$.
c. $\{(a, b)\}$.
e. $\{(a, a), (b, b), (c, c), (a, b)\}$.
g. $\{(a, a), (b, b), (c, c)\}$.

5. a. isGrandchildOf. **c.** isNephewOf.

6. isFatherOf $\circ$ isBrotherOf.

7. a. Let $R = \{(a, b), (b, a)\}$. Then R is irreflexive, and $R^2 = \{(a, a), (b, b)\}$, which is not irreflexive.

8. a. $\{(x, y) \mid x < y - 1\}$.

9. a. $\mathbb{N} \times \mathbb{N}$.
c. $\{(x, y) \mid y \neq 0\} - \{(0, 1)\}$.

11. $r(\varnothing) = \{(a, a) \mid a \in A\}$, which is basic equality over A.

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

13. a. $\varnothing$. **c.** $\{(a, b), (b, a), (a, a), (b, b)\}$.

15. a. isAncestorOf. **c.** greaterThan.

16. a.

	1	2	3	4
1	0	20	∞	5
2	∞	0	10	∞
3	∞	∞	0	10
4	∞	10	5	0

b.

	1	2	3	4
1	0	15	10	5
2	∞	0	10	20
3	∞	20	0	10
4	∞	10	5	0

	1	2	3	4
1	0	4	4	0
2	0	0	0	3
3	0	4	0	0
4	0	0	0	0

17. Let "path" be the function to compute the list of edges on a shortest path from i to j. We'll use the "cat" function to concatenate two lists:
path(i, j) = if $P_{ij} = 0$ then $\langle(i, j)\rangle$ else cat(path(i, P_{ij}), path(P_{ij}, j)).

19. Let M be the adjacency matrix for R. **a.** Check to see if $M_{ii} = 1$ for all i.
c. Check to see that "$M_{ij} = M_{jk} = 1$ implies $M_{ik} = 1$" for all i, j, and k.
e. For all i and j check M_{ij} . If $M_{ij} = 1$, then set $M_{ji} = 1$.

20. a. Let R be reflexive. Then $a\ R\ a$ and $a\ R\ a$ for all a, which implies that $a\ R^2\ a$ for all a. Therefore, R^2 is reflexive.
c. Let R be transitive, and let $a\ R^2\ b$ and $b\ R^2\ c$. Then $a\ R\ x$ and $x\ R\ b$, and $b\ R\ y$ and $y\ R\ c$ for some x and y. Since R is transitive, it follows that $a\ R\ b$ and $b\ R\ c$. Therefore, $a\ R^2\ c$. Thus R^2 is transitive.

21. Since less is transitive we have $t(\text{less}) = \text{less}$. It follows that $st(\text{less}) = s(\text{less}) = \{(m, n) \mid m \neq n\}$. But $ts(\text{less}) = t(\{(m, n) \mid m \neq n\}) = \mathbb{N} \times \mathbb{N}$.

22. a. $(x, y) \in R \circ (S \circ T)$ iff $(x, w) \in R$ and $(w, y) \in S \circ T$ for some w iff $(x, w) \in R$ and $(w, z) \in S$ and $(z, y) \in T$ for some w and z iff $(x, z) \in R \circ S$ and $(z, y) \in T$ for some z iff $(x, y) \in (R \circ S) \circ T$.
c. If $(x, y) \in R \circ (S \cap T)$, then $(x, w) \in R$ and $(w, y) \in S \cap T$ for some w. Thus $(x, y) \in R \circ S$ and $(x, y) \in R \circ T$, which implies that $(x, y) \in R \circ S \cap R \circ T$.

24. a. If R is reflexive, then it contains the set $\{(a, a) \mid a \in A\}$. Since $s(R)$ and $t(R)$ contain R as a subset, it follows that they each contain $\{(a, a) \mid a \in A\}$.
c. Suppose R is transitive. Let $(a, b), (b, c) \in r(R)$. If $a = b$ or $b = c$, then certainly $(a, c) \in r(R)$. So suppose $a \neq b$ and $b \neq c$. Then $(a, b), (b, c) \in R$. Since R is transitive, it follows that $(a, c) \in R$, which of course also says that $(a, c) \in r(R)$. Therefore, $r(R)$ is transitive.

25. a. A proof by containment goes as follows: If $(a, b) \in rt(R)$, then either $a = b$ or there is a sequence of elements $a = x_1, x_2, \ldots, x_n = b$ such that $(x_i, x_{i+1}) \in R$ for $1 \leq i < n$. Since $R \subset r(R)$, we also have $(x_i, x_{i+1}) \in r(R)$

for $1 \leq i < n$, which says that $(a, b) \in tr(R)$. For the other containment, let $(a, b) \in tr(R)$. If $a = b$, then $(a, b) \in rt(R)$. If $a \neq b$, then there is a sequence of elements $a = x_1, x_2, \ldots, x_n = b$ such that $(x_i, x_{i+1}) \in r(R)$ for $1 \leq i < n$. If $x_i = x_{i+1}$, then we can remove x_i from the sequence. So we can assume that $x_i \neq x_{i+1}$ for $1 \leq i < n$. Therefore, $(x_i, x_{i+1}) \in R$ for $1 \leq i < n$, which says that $(a, b) \in t(R)$, and thus also $(a, b) \in rt(R)$.
c. If $(a, b) \in st(R)$, then either $(a, b) \in t(R)$ or $(b, a) \in t(R)$. Without loss of generality we can assume that $(a, b) \in t(R)$. Then there is a sequence of elements $a = x_1, x_2, \ldots, x_n = b$ such that $(x_i, x_{i+1}) \in R$ for $1 \leq i < n$. Since $R \subset s(R)$, we also have $(x_i, x_{i+1}) \in s(R)$ for $1 \leq i < n$, which says that $(a, b) \in ts(R)$ (the symmetry also puts $(b, a) \in ts(R)$).

Section 4.2

1. a. Any point x is the same distance from the point as x. So $\sim$ is reflexive. If x and y are equidistant from the point, then y and x are too. So $\sim$ is symmetric. If x and y are equidistant from the point and y and z are equidistant from the point, then x, y, and z are equidistant from the point. Thus x and z are equidistant from the point. So $\sim$ is transitive.
c. $x + x$ is even for all natural numbers x. So $\sim$ is reflexive. If $x + y$ is even, then $y + x$ is even. So $\sim$ is symmetric. Let $x + y$ be even and let $y + z$ be even. Then $x + y = 2m$ and $y + z = 2n$ for some integers m and n. Solve for x and z to obtain $x = 2m - y$ and $z = 2n - y$. Add the two equations to obtain the equation $x + z = 2(m + n - y)$. Therefore, $x + z$ is even. So $\sim$ is transitive.
e. $xx > 0$ for all nonzero x. So $\sim$ is reflexive. If $xy > 0$, then $yx > 0$. So $\sim$ is symmetric. Let $xy > 0$ and $yz > 0$. Then either x and y are both positive or both negative. The same is true for y and z. So if y is positive, then x and z must be positive and if y is negative, then x and z must be negative. So in either case, we have $xz > 0$. So $\sim$ is transitive.

2. a. The relation is not reflexive because $a + a$ is always even. It is not transitive because, for example, $3 + 4$ is odd and $4 + 5$ is odd, but $3 + 5$ is not odd.
c. Not transitive. For example, $|2 - 7| \leq 5$ and $|7 - 12| \leq 5$, but $|2 - 12| > 5$.
e. Not reflexive: $(10, 10) \notin R$. Not symmetric: $(11, 10) \in R$, but $(10, 11) \notin R$.

3. a. $[0] = \mathbb{N}$.
c. $[2n] = \{2n, 2n + 1\}$ for each $n \in \mathbb{N}$.
e. $[4n] = \{4n, 4n + 1, 4n + 2, 4n + 3\}$ for each $n \in \mathbb{N}$.
g. $[0] = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$ and $[x] = \{x\}$ for $x \geq 12$.

4. a. $[x] = \{x, -x\}$ for $x \in \mathbb{Z}$.

5. a. Six classes $[0], [1], [2], [3], [4], [5]$, where $[n] = \{6k + n \mid k \in \mathbb{N}\}$.
c. Twelve classes $[0], [1], [2], \ldots, [11]$, where $[n] = \{12k + n \mid k \in \mathbb{N}\}$.

6. a. Two classes $\{rot, roto, root\}$ and $\{tot, toot, toto, too, to, otto\}$.

7. a. The weight is 7. One answer is $\{\{a, c\}, \{e, c\}, \{c, d\}, \{b, d\}, \{d, f\}\}$.

9. Let $x \in A$. Since E and F are reflexive, we have $(x, x) \in E$ and $(x, x) \in F$, so it follows that $(x, x) \in E \cap F$. Thus $E \cap F$ is reflexive. Let $(x, y) \in E \cap F$. Then $(x, y) \in E$ and $(x, y) \in F$. Since E and F are symmetric, we have $(y, x) \in E$ and $(y, x) \in F$. So $(y, x) \in E \cap F$. Thus $E \cap F$ is symmetric. Let (x, y), $(y, z) \in E \cap F$. Then (x, y), $(y, z) \in E$ and (x, y), $(y, z) \in F$. Since E and F are transitive it follows that $(x, z) \in E$ and $(x, z) \in F$. So $(x, z) \in E \cap F$. Thus $E \cap F$ is transitive, and hence an equivalence relation on A.

11. $tsr(R) = trs(R) = rts(R)$ and $str(R) = srt(R) = rst(R)$.

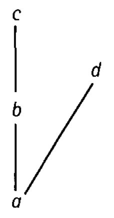
13. The function $s : A \to P$ defined by $s(a) = [a]$ is a surjection because every element in P has the form $[a]$ for some $a \in A$. The function $i : P \to B$ defined by $i([a]) = f(a)$ is an injection because if $i([a]) = i([b])$, then $f(a) = f(b)$. But this says that $[a] = [b]$, which implies that i is injective. To see that $f = i \circ s$, notice that $(i \circ s)(a) = i(s(a)) = i([a]) = f(a)$.

Section 4.3

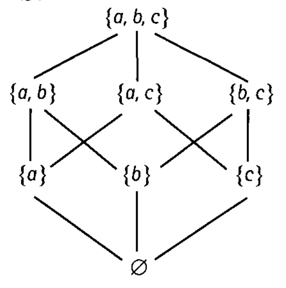
1. **a.** False.
c. True.

2. **a.** No.
c. Yes.
e. No.

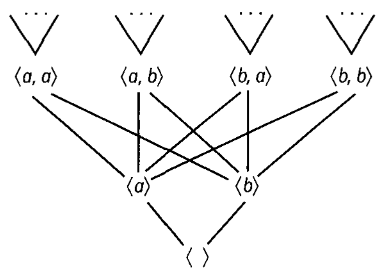
3. **a.**



b.



c.



5. The glb of two elements is their greatest common divisor, and the lub is their least common multiple.

7. **a.** No tree has fewer than zero nodes. Therefore, every descending chain of trees is finite if the order is by the number of nodes.
c. No list has length less than zero. Therefore, every descending chain of lists is finite if the order is by the length of the list.

9. Yes.

11. An element i is a source if the ith column is full of 0's. When a source i is output, set the elements in the ith row to 0.

13. Suppose A is well-founded and S is a nonempty subset of A. If S does not have a minimal element, then there is an infinite descending chain of elements in S, which contradicts the assumption that A is well-founded. For the converse, suppose that every nonempty subset of A has a minimal element. So any descending chain of elements from A is a nonempty subset of A that must have a minimal element. Thus the descending chain must be finite. Therefore, A is well-founded.

14. a. Yes.
c. No. For example, $-2 < 1$, but $f(-2) > f(1)$.
e. Yes.
g. No.

Section 4.4

1. 2900.

2. a. The equation is true if $n = 1$. So assume that the equation is true for n, and prove that it's true for $n + 1$. Starting on the left-hand side, we get

$$\begin{aligned}1+3+\cdots+(2n-1)+(2(n+1)-1) &= (1+3+\cdots+(2n-1)) \\ &\quad +(2(n+1)-1) \\ &= n^2+(2(n+1)-1) \\ &= n^2+2n+1=(n+1)^2.\end{aligned}$$

c. The equation is true for $n = 1$. So assume that the equation is true for n, and prove that it's true for $n + 1$. Starting on the left-hand side, we get

$$\begin{aligned}3+7+11+\cdots+(4n-1)+[4(n+1)-1] &= (3+7+11+\cdots+(4n-1)) \\ &\quad +[4(n+1)-1] \\ &= n(2n+1)+4(n+1)-1 \\ &= 2n^2+n+4n+3 \\ &= 2n^2+5n+3 \\ &= (n+1)(2n+3) \\ &= (n+1)(2(n+1)+1).\end{aligned}$$

e. The equation is true for $n = 0$. So assume that the equation is true for n, and prove that it's true for $n + 1$. Starting on the left-hand side, we get

$$\begin{aligned}1+5+9+\cdots+(4n+1)+[4(n+1)+1] &= 1+5+9+\cdots+(4n+1) \\ &\quad +[4(n+1)+1] \\ &= (n+1)(2n+1)+4(n+1)+1 \\ &= 2n^2+3n+1+4n+5 \\ &= 2n^2+7n+6 \\ &= (2n+3)(n+2) \\ &= (2(n+1)+1)((n+1)+1).\end{aligned}$$

g. The equation is true for $n = 1$. So assume that the equation is true for n, and prove that it's true for $n + 1$. Starting with the left side of the equation for $n + 1$, we get

$$\begin{aligned}1^2 + 2^2 + \cdots + (n+1)^2 &= \left(1^2 + 2^2 + \cdots + n^2\right) + (n+1)^2 \\ &= \frac{n(n+1)(2n+1)}{6} + (n+1)^2 \\ &= \frac{(n+1)((n+1)+1)(2(n+1)+1)}{6}.\end{aligned}$$

i. The equation is true for $n = 2$. So assume that the equation is true for n, and prove that it's true for $n + 1$. Starting on the left-hand side, we get

$$\begin{aligned}&2 + 6 + 12 + \cdots \\ &\quad + \left[(n+1)^2 - (n+1)\right] = 2 + 6 + 12 + \cdots + \left(n^2 - n\right) \\ &\qquad\qquad + \left[(n+1)^2 - (n+1)\right] \\ &\quad = \frac{n\left(n^2 - 1\right)}{3} + \left[(n+1)^2 - (n+1)\right] \\ &\quad = \frac{n(n+1)(n-1) + 3\left[(n+1)^2 - (n+1)\right]}{3} \\ &\quad = \frac{(n+1)\left[n(n-1) + 3(n+1) - 3\right]}{3} \\ &\quad = \frac{(n+1)\left[n^2 + 2n\right]}{3} \\ &\quad = \frac{(n+1)\left[\left(n^2 + 2n + 1\right) - 1\right]}{3} \\ &\quad = \frac{(n+1)\left[(n+1)^2 - 1\right]}{3}.\end{aligned}$$

3. a. For $n = 1$ the equation becomes $0 = 1 - 1$. Assume that the equation is true for n. Then the case for $n + 1$ goes as follows:

$$\begin{aligned}F_0 + F_1 + \cdots + F_n + F_{n+1} &= (F_0 + F_1 + \cdots + F_n) + F_{n+1} \\ &= F_{n+2} - 1 + F_{n+1} = F_{n+3} - 1.\end{aligned}$$

c. For $(m, n) = (0, 0)$ the equation becomes $0 = 0 + 0$. Assume that the equation is true for all $(i, j) \prec (m, n)$. Then the case for (m, n) goes as follows:

$$\begin{aligned}F_{m+n} &= F_{m+n-1} + F_{m+n-2} \\ &= (F_{m+1}F_{n-1} + F_mF_n) + (F_{m+1}F_{n-2} + F_mF_{n-1}) \\ &= F_{m+1}(F_{n-1} + F_{n-2}) + F_m(F_n + F_{n-1}) \\ &= F_{m+1}F_n + F_mF_{n+1}.\end{aligned}$$

4. a. For $n = 0$ the equation becomes $2 = 3 - 1$. Assume that the equation is true for n. Then the case for $n + 1$ goes as follows:

$$\begin{aligned} L_0 + L_1 + \cdots + L_n + L_{n+1} &= (L_0 + L_1 + \cdots + L_n) + L_{n+1} \\ &= L_{n+2} - 1 + L_{n+1} = L_{n+3} - 1. \end{aligned}$$

5. Let $P(m, n)$ denote the equation. Induct on the variable n. For any m we have $\text{sum}(m + 0) = \text{sum}(m) = \text{sum}(m) + \text{sum}(0) + m0$. So $P(m, 0)$ is true for arbitrary m. Now assume that $P(m, n)$ is true, and prove that $P(m, n + 1)$ is true. Starting on the left-hand side we get

$$\begin{aligned} \text{sum}\,(m + (n + 1)) &= \text{sum}\,((m + n) + 1) \\ &= \text{sum}\,(m + n) + m + n + 1 \\ &= \text{sum}\,(m) + \text{sum}\,(n) + mn + m + n + 1 \\ &= \text{sum}\,(m) + \text{sum}\,(n + 1) + m\,(n + 1)\,. \end{aligned}$$

Therefore, $P(m, n + 1)$ is true. Therefore, $P(m, n)$ is true for all m and n.

7. Since L is transitive and $R \subset L$, it follows that $t(R) \subset L$. For the other direction, let $(x, y) \in L$. In other words, $x < y$. Therefore, there is some natural number $k \geq 1$ such that $y = x + k$. We'll induct on k. If $k = 1$ then we have $(x, y) = (x, x + 1) \in R$. So $(x, y) \in t(R)$. Now assume that $(x, x + k) \in t(R)$ and prove $(x, x + k + 1) \in t(R)$. Since $(x, x + k) \in t(R)$ and $(x + k, x + k + 1) \in R \subset t(R)$, it follows by the transitivity of $t(R)$ that $(x, x + k + 1) \in t(R)$. Therefore, $(x, y) \in t(R)$. Therefore, $L \subset t(R)$. So we have $t(R) = L$.

9. For lists K and L, let $K \prec L$ mean the length of K is less than than the length of L. This forms a well-founded ordering on lists. Let $P(L)$ denote the statement "$f(L)$ is the length of L." Notice that $f(\langle\ \rangle) = 0$. Therefore, $P(\langle\ \rangle)$ is true. Now let $L \neq \langle\ \rangle$ and assume $P(K)$ is true for all lists $K \prec L$. In other words, we are assuming "$f(K)$ is the length of K" for all $K \prec L$. We must show $P(L)$ is true. Since $L \neq \langle\ \rangle$, we have $f(L) = 1 + f(\text{tail}(L))$. Since $\text{tail}(L) \prec L$, our induction assumption applies and we have $P(\text{tail}(L))$ is true. In other words, $f(\text{tail}(L))$ is the length of $\text{tail}(L)$. Thus $f(L)$ is 1 plus the length of $\text{tail}(L)$, which of course is the length of L.

10. a. Let T be a binary tree. We know that an empty tree has no nodes. Since $g(\langle\ \rangle) = 0$, we know that the function is correct when $T = \langle\ \rangle$. For the induction part we need a well-founded ordering on binary trees. For example, let $t \prec s$ mean that t is a subtree of s. Now assume that T is a nonempty binary tree, and also assume that the function is correct for all subtrees of T. Since T is nonempty, it has the form $T = \text{tree}(L, x, R)$. We know that the number of nodes in T is equal to the number of nodes in L plus those in R plus 1. The function g, when given argument T, returns $1 + g(L) + g(R)$. Since L and R are subtrees of T, it follows by assumption that $g(L)$ and $g(R)$ represent the number of nodes in L and R, respectively. Thus $g(T)$ is the number of nodes in T.

11. a. If $L = \langle x \rangle$, then $\text{forward}(L) = \{\text{print}(\text{head}(L)); \text{forward}(\text{tail}(L))\} = \{\text{print}(x); \text{forward}(\langle\ \rangle)\} = \{\text{print}(x)\}$. We'll use the well-founded ordering based on the length of lists. Let L be a list with n elements, where $n > 1$, and assume that forward is correct for all lists with fewer than n elements. Then forward(L) = {print(head(L)); forward(tail(L))}. Since tail(L) has fewer than n elements, forward(tail(L)) correctly prints out the elements of tail(L) in the order listed. Since print(head(L)) is executed before forward(tail(L)), it follows that forward(L) is correct.

12. a. We can use well-founded induction, where $L \prec M$ if length(L) < length(M). Since an empty list is sorted and sort($\langle\ \rangle$) = $\langle\ \rangle$, it follows that the function is correct for the basis case $\langle\ \rangle$. For the induction case, assume that sort(L) is sorted for all lists L of length n, and show that sort($x :: L$) is sorted. By definition, we have sort($x :: L$) = insert(x, sort(L)). The induction assumption implies that sort(L) is sorted. Therefore, insert(x, sort(L)) is sorted by the assumption in the problem. Thus sort($x :: L$)) is sorted.

13. Let's define the following order on pairs of positive integers: $(a, b) \prec (c, d)$ iff $a < c$ and $b \le d$, or $a \le c$ and $b < d$. This is a well-founded ordering with least element (1, 1). For the base case we have $g(1, 1) = 1$. So g is correct in this case because gcd(1, 1) = 1. For the induction case, assume (x, y) is a pair of positive integers and assume $g(x', y') = \gcd(x', y')$ for all $(x', y') \prec (x, y)$. If $x = y$ then of course $g(x, y) = x = \gcd(x, y)$. So assume $x < y$. Then $g(x, y) = g(x, y - x)$. Since $(x, y - x) \prec (x, y)$ the induction assumption says that $g(x, y - x) = \gcd(x, y - x)$. Since $\gcd(x, y) = \gcd(x, y - x)$ (by (2.1b)), it follows that $g(x, y) = \gcd(x, y)$. The argument is similar if $y < x$.

15. We'll induct on the list variable. So we need a well-founded ordering on lists. For lists L and M, let $L \prec M$ mean length(L) < length(M). Let $P(x, L)$ be the statement, "delete(x, L) returns L with the first occurrence of x deleted." We need to show that $P(x, L)$ is true for all lists L. The single minimal element is $\langle\ \rangle$. The definition of delete gives delete(x, $\langle\ \rangle$) = $\langle\ \rangle$. This makes sense because $\langle\ \rangle$ is the result of deleting x from $\langle\ \rangle$. Therefore, the base case $P(x, \langle\ \rangle)$ is true. For the induction case, let K be a nonempty list and assume $P(x, L)$ is true for all $L \prec K$. We need to show $P(x, K)$ is true. There are two cases. The first case is x = head(K). Then delete(x, K) = tail(K), which is clearly the result of removing the first occurrrence of x from K. Therefore, $P(x, K)$ is true. Now assume $x \ne$ head(K). Then the definition of delete gives

$$\text{delete}(x, K) = \text{head(K)} :: \text{delete}(x, \text{tail}(K)).$$

Since tail(K) $\prec K$, the induction assumption says that $P(x$, tail(K)) is true. Therefore, $P(x, K)$ is true because the first element of delete(x, K) is not equal to x. Therefore, (4.28) applies to say delete(x, L) is true for all lists L.

17. If $a = b$, then the equation holds. So assume $a \ne b$. The equation holds for $L = \langle\ \rangle$. Assume $L = x :: M$ and assume the equation holds for all lists having length less than that of L. Then the left side of the equation becomes $r(a, r(b, x :: M)$. If $x = b$, then the expression becomes $r(a, r(b, b :: M))$. But $r(b, b :: M) = r(b, M)$. Therefore, the left side becomes $r(a, r(b, M))$. The induction assumption then allows us to write this expression as $r(b, r(a, M))$. Now look at the right side of the equation. We have $r(b,\ r(a,\ x :: M))$. Still assuming $x = b$, we write $r(b,\ r(a,\ b :: M)) = r(b,\ b :: r(a,\ M)) = r(b,\ r(a,\ M))$. Thus the equation holds if $x = b$. A similar argument tells us that the equation holds if $x = a$. Lastly, assume $x \neq a$ and $x \neq b$. Then we can write $r(a,\ r(b,\ x :: M))$ $= r(a,\ x :: r(b,\ M)) = x :: r(a,\ r(b,\ M))$. Now apply the induction assumption to the last expression to get $x :: r(b,\ r(a,\ M))$. But we can reach this expression if we start on the right side: $r(b,\ r(a,\ x :: M)) = r(b,\ x :: r(a,\ M)) = x :: r(b,\ r(a,\ M))$. Thus the equation is true for any list.

19. a. If $L = \langle\ \rangle$, then isMember(a, L) = false, which is correct. Now assume that L has length n and that isMember(a, K) is correct for all lists K of length less than n. If a = head(L), then isMember(a, L) = true, which is correct. So assume that $a \neq$ head(L). It follows that $a \in L$ iff $a \in$ tail(L). Since $a \neq$ head(L), it follows that isMember(a, L) = isMember(a, tail(L)). Since tail(L) has fewer than n elements, the induction assumption says that isMember(a, tail(L)) is correct. Therefore, isMember(a, L) is correct for any list L.

20. If $x = \langle\ \rangle$, then the definition of cat implies that cat($\langle\ \rangle$, cat(y, z)) = cat(y, z) = cat(cat($\langle\ \rangle$, y), z). Now assume that the statement is true for x, and prove the statement for $a :: x$:

$$\begin{aligned}
\text{cat}(a :: x,\ \text{cat}(y, z)) &= a :: \text{cat}(x,\ \text{cat}(y, z)) && \text{(definition)}\\
&= a :: \text{cat}(\text{cat}(x, y)\,, z) && \text{(induction)}\\
&= \text{cat}(a :: \text{cat}(x, y)), z) && \text{(definition)}\\
&= \text{cat}(\text{cat}(a :: x, y)\,, z) && \text{(defintion).}
\end{aligned}$$

So the statement is true for $a :: x$ under the assumption that it is true for x. It follows by structural induction that the statement is true for all x, y, and z.

22. Let W be a well-founded set, and let S be a nonempty subset of W. We'll assume condition 2 of (4.27): Whenever an element x in W has the property that all its predecessors are elements in S, then x also is an element in S. We want to prove condition 1 of (4.27): S contains all the minimal elements of W. Suppose, by way of contradiction, that there is some minimal element $x \in W$ such that $x \notin S$. Then all predecessors of x are in S because there aren't any predecessors of x. Condition 2 of (4.27) now forces us to conclude that $x \in S$, a contradiction. Therefore, condition 1 of (4.27) follows from condition 2 of (4.27).

24. a. If we can show that $f(n,\ 0,\ 1) = f(k,\ F_{n-k},\ F_{n-k+1})$ for all $0 \leq k \leq n$, then for $k = 0$ we have $f(n,\ 0,\ 1) = f(0,\ F_n,\ F_{n+1}) = F_n$, by the definition of f. To prove that $f(n,\ 0,\ 1) = f(k,\ F_{n-k},\ F_{n-k+1})$ for all $0 \leq k \leq n$, we'll fix n and induct on the variable k as it ranges from n down to 0. So the basis case is $k = n$. In this case we have

$$f(n,\ 0,\ 1) = f(n,\ F_0,\ F_1) = f(k,\ F_{n-k},\ F_{n-k+1}).$$

For the induction case, assume that $f(n,\ 0,\ 1) = f(k,\ F_{n-k},\ F_{n-k+1})$ for some k such that $0 < k \leq n$, and prove that

$$f(n,\ 0,\ 1) = f(k-1,\ F_{n-k+1},\ F_{n-k+2}).$$

We have the following equations:

$$\begin{aligned} f(n,0,1) &= f(k, F_{n-k}, F_{n-k+1}) && \text{(induction assumption)} \\ &= f(k-1, F_{n-k+1}, F_{n-k} + F_{n-k+1}) && \text{(definition of } f\text{)} \\ &= f(k-1, F_{n-k+1}, F_{n-k+2}). && \text{(definition of } F_{n-k+2}\text{).} \end{aligned}$$

Therefore, $f(n, 0, 1) = f(k, F_{n-k}, F_{n-k+1})$ for all $0 \le k \le n$.

25. Both formulas give $d(2) = 1$. For the induction part let $n > 2$ and assume that $d(k) = kd(k-1) + (-1)^k$ for $k < n$. Show that $d(n) = nd(n-1) + (-1)^n$. Using the original formula we have

$$\begin{aligned} d(n) &= (n-1)(d(n-1) + d(n-2)) \\ &= nd(n-1) - d(n-1) + nd(n-2) - d(n-2) \end{aligned}$$

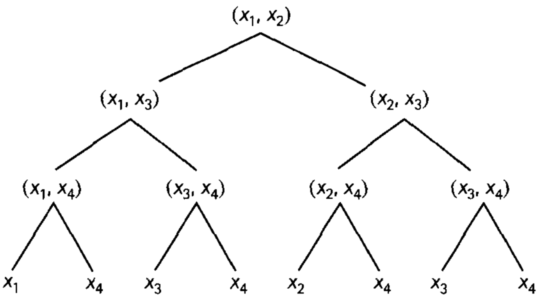
Now use the hypothesis to replace the second occurrence of $d(n-1)$ to obtain

$$\begin{aligned} &= nd(n-1) - \left[(n-1)d(n-2) + (-1)^{n-1}\right] + nd(n-2) - d(n-2) \\ &= nd(n-1) - nd(n-2) + d(n-2) - (-1)^{n-1} + nd(n-2) - d(n-2) \\ &= nd(n-1) - (-1)^{n-1} = nd(n-1) + (-1)^n. \end{aligned}$$

Chapter 5

Section 5.1

1. In the following tree, move to the left child of (a, b) whenever $a > b$.



2. a. 7.
c. 4.

3. There are 61 possible outcomes. So there must be at least 61 leaves on any tree that solves the problem. If h is the height of a ternary decision tree, then the tree has at most 3^h leaves. Therefore, $61 \le 3^h$. Take the log to conclude that $h \ge \text{ceiling}(\log_3(61)) = 4$. So 4 is a reasonable lower bound.

Section 5.2

1. a. $2(1) + 3 + 2(2) + 3 + 2(3) + 3 + 2(4) + 3 + 2(5) + 3$.
c. $5(3^0) + 4(3^1) + 3(3^2) + 2(3^3) + 1(3^4)$.

2. a. $\sum_{i=0}^{n-1} g(i)\, a_{i+1} x^{i+2}$.

c. $\sum_{i=-1}^{n-2} g(i+1)\, a_{i+2} x^{i+3}$.

e. $\sum_{i=-2}^{n-3} g(i+2)\, a_{i+3} x^{1+4}$.

3. a. $\sum_{i=1}^{n} 3i = 3\sum_{i=1}^{n} i = \frac{3n(n+1)}{2}$.

c. $\sum_{i=0}^{n} 3(2^i) = 3\sum_{i=0}^{n} 2^i = 3(2^{n+1} - 1)$.

4. a. $n^2 + 3n$.
c. $n^2 + 4n$.
e. $2n^2$.
g. $\frac{n(n+1)(n+2)}{3}$.

5. $(1/4)n^2(n + 1)^2$.

6. a. $n^2 + 3n$.

7. a. $2 + 3 + \cdots + (\text{n}+ 1) = (1/2)(n + 1)(n + 2) - 1$.

8. a. $3+5+\ldots+(2k+3)$, where $k = \text{ceiling}(n/2) - 1$. This sum has the value $(k+1)(k+3) = (k+2)^2 - 1 = (\text{ceiling}(n/2) + 1)^2 - 1$.

Section 5.3

1. a. 720.
c. 30.
e. 10.

2. a. *abc*, *acb*, *bac*, *bca*, *cab*, *cba*. **c.** $\{a, b, c\}$.
e. $[a, a]$, $[a, b]$, $[a, c]$, $[b, b]$, $[b, c]$, $[c, c]$.

3. a. $P(3, 3)$.
c. $C(3, 3)$.
e. $C(3 + 2 - 1, 2)$.

4. The number of bag permutations of B is $4!/(2!2!) = 6$. They can be listed as follows: *aabb*, *abab*, *abba*, *bbaa*, *baba*, *baab*.

5. a. $8! = 40{,}320$.
c. $6!/(2!2!1!1!) = 180$.
e. $9!/(4!2!2!1!) = 3780$.

6. a. There are none.
c. *BCA*, *CAB*.

7. $n = 7$, $k = 3$, and $m = 4$.

9. Either $\text{floor}(n/2) + 1$ or $\text{ceiling}(n/2) + 1$.

10. There are eight possible outcomes when three coins are tossed.
a. 0.375.
c. 0.875.

11. There are 36 possible outcomes.
a. 0.1666... .
c. 0.222... .

12. **a.** 80/243.
c. 4/9.

13. **a.** 0.21.

15. 1/3, 2/15, 8/15.

16. **a.** 1/2.
c. no.

18. **a.** 1C(49, 6), since order is not important.
c. $[C(6,5)\,C(43,1) + C(6,4)\,C(43,2)]\,/C(49,6)$. To obtain the answer, notice that there are $C(49,6)$ 6-element subsets of a 49-element set. Now suppose that $\{a,b,c,d,e,f\}$ is the winning set of numbers. First, we'll count all the 6-element sets that contain exactly five winners. To do this, notice that there are $C(6,5)$ 5-element subsets of $\{a,b,c,d,e,f\}$. To each of these 5-element subsets we add a non-winner from the set $\{1,\ldots,49\} - \{a,b,c,d,e,f\}$. Since there are 43 $(= C(43,1))$ non-winners, it follows that there are $C(6,5)\,C(43,1)$ sets of 6 numbers that contain exactly five winners. Next, we'll count the 6-element sets that contain exactly four winners. To do this, notice that there are $C(6,4)$ 4-element subsets of $\{a,b,c,d,e,f\}$. To each of these 4-element subsets we add two non-winners from the set $\{1,\ldots,49\} - \{a,b,c,e,f\}$. Since there are $C(43,2)$ possible pairs of non-winners, it follows that there are $C(6,4)\,C(43,2)$ sets of six numbers that contain exactly four winners. So the probability of choosing either four or five of the six winning numbers is given by $[C(6,5)\,C(43,1) + C(6,4)\,C(43,2)]\,/$ $C(49,6)$.

19. **a.** 3.5.

21. Since A is a subset S we have $A \cap S = A$. So $P(A \cap S) = P(A) = P(A)P(S)$. Thus S and A are independent. If $A \cap B = \varnothing$, then the only way for A and B to be independent is if $A = B = \varnothing$.

23. There are nk actions to schedule. Since the k actions of each process must be done in order, we can represent each process as a bag consisting of k identical elements. Assume that the bags are disjoint from each other. Then the union B of the n bags contains nk elements, and each bag permuation of B is one schedule. Therefore, there are as many schedules as there are bag permutations of B. That number is $(nk)!/(k!)^n$.

25. Let $\mid S \mid = n$. If $n = 1$, then there is just one bijection, the identity mapping on S. Let $n > 1$ and assume there are $k!$ bijections between any two sets with k elements when $k < n$. Pick some element $x \in S$. Any bijection of S must map x to one of its n elements y. The remaining elements in $S - \{x\}$ must be mapped to $S - \{y\}$. These sets each have $n - 1$ elements. The induction assumption tells us there are $(n - 1)!$ bijections from $S - \{x\}$ to $S - \{y\}$. Therefore, there are $n(n - 1)! = n!$ bijections from S to S.

26. a. 109/30 (or about 3.63).
b. $(p/n)[(n + 1)\log_2(n + 1) - n] + (1 - p)\log_2(n + 1)$.

Section 5.4

1. a. $4(n - 1)$.
c. $2^{n+2} - 3$.

2. a. Let a_n be the number of cons operations when L has length n. Then $a_0 = 0$ and $a_n = 1 + a_{n-1}$, which has solution $a_n = n$. **c.** Let a_n be the number of cons operations when L has length n. Then $a_0 = 1$, and $a_n = a_{n-1} + 5 \cdot 2^{n-1}$, which has solution $a_n = 5 \cdot 2^n - 4$.

3. The recurrence is given by $H_0 = 0$ and $H_n = 2H_{n-1} + 1$. The solution is $H_n = 2^n - 1$.

5. Let r_n be the number of regions created by n lines. Then $r_0 = 1$ since a plane with no lines is one region. It's easy to see that $r_1 = 2$, $r_2 = 4$, $r_3 = 7$, and $r_4 = 11$. After some thought, we see that when there are $n - 1$ lines in the plane and we add one more line, it intersects each of the existing $n - 1$ lines and splits up n existing regions. So $r_n = r_{n-1} + n$. This recurrence can be solved by substitution or cancellation to get $r_n = (1/2)(n^2 + n + 2)$.

6. a. $a_n = -1/(2^{n+1}) - 2(-3)^n$.
c. $a_n = (-1/2)(3/2)^n - n - 1$.

7. a. $a_n = 3^n + (-1)^{n+1}$.
c. $a_n = (1/3)(2^n - (-1)^n)$.

8. a. $A(x) = 4/(1 - x)^2 - 8/(1 - x) + 4$, which yields $a_n = 4(n - 1)$.
c. $A(x) = -3/(1 - x) + 4/(1 - 2x)$, which yields $a_n = 2^{n+2} - 3$.

9. a. If $n = 1$, then the equation evaluates to $2 = 2$. Assume that the equation holds for n, and show that it holds for $n + 1$. Starting with the left side for the $n + 1$ case we have

$$\begin{aligned}
&\frac{(1)(1)(3)(5)\cdots(2n-3)(2n-1)}{(n+1)!}2^{n+1}\\
&= \frac{(1)(1)(3)(5)\cdots(2n-3)}{n!}2^n\frac{2n-1}{n+1}2\\
&= \frac{2}{n}\binom{2n-2}{n-1}\frac{2n-1}{n+1}2 = \frac{2}{n+1}\binom{2(n+1)-2}{n+1-1}
\end{aligned}$$

which is the right side for the $n + 1$ case.

10. Letting $F(x)$ be the generating function for F_n, we get $F(x) = x/(1 - x - x^2)$. The denominator factors into $1 - x - x^2 = (1 - \alpha x)(1 - \beta x)$, where

$$\alpha = \frac{1}{2}\left(1 + \sqrt{5}\right) \quad \text{and} \quad \beta = \frac{1}{2}\left(1 - \sqrt{5}\right).$$

Now use partial fractions to obtain

$$F(x) = \frac{1}{\sqrt{5}}\left(\frac{1}{1-\alpha x} - \frac{1}{1-\beta x}\right).$$

This yields the closed formula $F_n = \frac{1}{\sqrt{5}}(\alpha^n - \beta^n)$.

Section 5.5

1. $1 \prec \log\log n \prec \log n$.

2. a. $f(n) = \Theta(n \log n)$. Notice that $f(n) = \log(1 \cdot 2 \cdot \cdots \cdot n) = \log(n!)$. Now use (5.37) to approximate $n!$ and take the log of Stirlings's formula to obtain $\Theta(n \log n)$.

3. a. $5! = 120$; Stirling ≈ 118.02; diff $= 1.98$.

4. Let $f(n) = (n-1)/n$. Then f is increasing, and for all $n \geq 2$ we have the inequality $(1/2) \cdot 1 \leq f(n) \leq 1 \cdot 1$. Therefore, $f(n) = \Theta(1)$.

5. (Reflexive) Since $1|f(n)| \leq |f(n)| \leq 1|f(n)|$, it follows that $f(n) = \Theta(f(n))$ for every functon f. (Symmetric) If $f(n) = \Theta(g(n))$, then there are positive constants, c, d, and a number m such that $c|g(n)| \leq |f(n)| \leq d|g(n)|$ for all $n \geq m$. It follows that $(1/d)|f(n)| \leq |g(n)| \leq (1/c)|f(n)|$ for all $n \geq m$. Thus $g(n) = \Theta(f(n))$. (Transitive) Assume that $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$. Then there are positive constants c, d, and a number m such that $c|g(n)| \leq |f(n)| \leq d|g(n)|$ for $n \geq m$. Similarly, there are positive constants a, b, and a number k such that and $a|h(n)| \leq |g(n)| \leq b|h(n)|$ for $n \geq k$. It follows that $(ca)|h(n)| \leq |f(n)| \leq (bd)|h(n)|$ for $n \geq \max\{m, k\}$. This says that $f(n) = \Theta(h(n))$.

6. a. The quotient $\log(kn)/\log n$ approaches 1 as n approaches infinity.

8. Take limits.

9. In each case, replace $n!$ by its Stirling's approximation (5.37). Then take limits.

10. a. Since $f(n) = O(h(n))$, there are positive constants c and m such that $| f(n) | \leq c \mid g(n) \mid$ for all $n \geq m$. If $a = 0$ then $af(n) = 0$ for all n. So $af(n) = O(h(n))$. If $a \neq 0$, then multiply both sides of the inequality by $| a |$ to obtain $| af(n) | \leq (c \mid a \mid) \mid g(n) \mid$ for all $n \geq m$. Thus $af(n) = O(h(n))$.

Chapter 6

Section 6.2

1. a. $((\neg P) \wedge Q) \rightarrow (P \vee R)$.
c. $(A \rightarrow (B \vee (((\neg C) \wedge D) \wedge E))) \rightarrow F$.

2. a. $(P \vee Q \rightarrow \neg R) \vee \neg Q \wedge R \wedge P$.

3. $(A \rightarrow B) \wedge (\neg A \rightarrow C)$ or $(A \wedge B) \vee (\neg A \wedge C)$.

5. $A \wedge \neg B \to \text{false} \equiv \neg(A \wedge \neg B) \vee \text{false} \equiv \neg(A \wedge \neg B) \equiv \neg A \vee \neg\neg B \equiv \neg A \vee B \equiv A \to B$.

7. a. If B = true, then the wff is true. If B = false and A = true, the wff is false.
c. If A is true, then the wff is true. If A = false and C = true, the wff is false.
e. If B = true, then the wff is true. If B = false, $A = C$ = true, the wff is false.

8. a. If C = true, $A \to C$ is true, so the wff is trivially true too. If C = false, then the wff becomes $(A \to B) \wedge (B \to \text{false}) \to (A \to \text{false})$, which is equivalent to $(A \to B) \wedge \neg B \to \neg A$. If A = false, then the wff is trivially true. If A = true, the wff becomes

$$(\text{true} \to B) \wedge \neg B \to \text{false} \equiv B \wedge \neg B \to \text{false} \equiv \text{false} \to \text{false} \equiv \text{true}.$$

c. If A = false or B = false, then the consequent is true, so the statement is trivially true. If $A = B$ = true, then the wff becomes

$$(\text{true} \to C) \wedge (\text{true} \to D) \wedge (\neg C \vee \neg D) \to \text{false}$$
$$\equiv C \wedge D \wedge (\neg C \vee \neg D) \to \text{false}.$$

If C = true, then the wff becomes

$$\text{true} \wedge D \wedge (\text{false} \vee \neg D) \to \text{false} \equiv D \wedge \neg D \to \text{false} \equiv \text{false} \to \text{false} \equiv \text{true}.$$

If C = false, then the wff becomes

$$\text{false} \wedge D \wedge (\text{true} \vee \neg D) \to \text{false} \equiv \text{false} \to \text{false} \equiv \text{true}.$$

e. If B is true, then the wff is trivially true. If B is false, then the wff becomes

$$(\text{true} \to \neg A) \to ((\text{true} \to A) \to \text{false}) \equiv \neg A \to (A \to \text{false})$$
$$\equiv \neg A \to \neg A \equiv \text{true}.$$

g. If C is true, then the wff is trivially true. If C is false, then the wff becomes $(A \to \text{false}) \to ((B \to \text{false}) \to (A \vee B \to \text{false})) \equiv \neg A \to (\neg B \to \neg(A \vee B))$. If A is true, then the wff is vacuously true. If A if false, then the wff becomes

$$\text{true} \to (\neg B \to \neg(\text{false} \vee B)) \equiv \neg B \to \neg(\text{false} \vee B) \equiv \neg B \to \neg B \equiv \text{true}.$$

9. a. $(A \to B) \wedge (A \vee B) \equiv (\neg A \vee B) \wedge (A \vee B) \equiv (\neg A \wedge A) \vee B$
$\equiv \text{false} \vee B \equiv B$.
c. $A \wedge B \to C \equiv \neg(A \wedge B) \vee C \equiv (\neg A \vee \neg B) \vee C \equiv \neg A \vee (\neg B \vee C)$
$\equiv \neg A \vee (B \to C) \equiv A \to (B \to C)$.
e. $A \to B \wedge C \equiv \neg A \vee (B \wedge C) \equiv (\neg A \vee B) \wedge (\neg A \vee C)$
$\equiv (A \to B) \wedge (A \to C)$.

10. a. $A \to A \vee B \equiv \neg A \vee A \vee B \equiv \text{true} \vee B \equiv \text{true}$.
c. $(A \vee B) \wedge \neg A \to B \equiv \neg((A \vee B) \wedge \neg A) \vee B \equiv \neg(A \vee B) \vee A \vee B \equiv (\neg A \wedge \neg B) \vee (A \vee B) \equiv (\neg A \vee A \vee B) \wedge (\neg B \vee A \vee B) \equiv (\text{true} \vee \neg B) \wedge (\text{true} \vee A) \equiv \text{true} \wedge \text{true} \equiv \text{true}$.

e. $(A \to B) \wedge \neg B \to \neg A \equiv (\neg A \vee B) \wedge \neg B \to \neg A \equiv \neg((\neg A \vee B) \wedge \neg B) \vee \neg A \equiv (A \wedge \neg B) \vee (B \vee \neg A) \equiv (A \vee B \vee \neg A) \wedge (\neg B \vee B \vee \neg A)$ $\equiv (\text{true} \vee B) \wedge (\text{true} \vee \neg A) \equiv \text{true} \wedge \text{true} \equiv \text{true}$.
g. $A \to (B \to (A \wedge B)) \equiv \neg A \vee (\neg B \vee (A \wedge B)) \equiv (\neg A \vee \neg B) \vee (A \wedge B)$ $\equiv \neg (A \wedge B) \vee (A \wedge B) \equiv \text{true}$.

11. **a.** $(P \wedge \neg Q) \vee P$ or P. **c.** $\neg Q \vee P$. **e.** $\neg P \vee (Q \wedge R)$.

12. **a.** $P \wedge (\neg Q \vee P)$ or P. **c.** $\neg Q \vee P$. **e.** $(\neg P \vee Q) \wedge (\neg P \vee R)$.
g. $(A \vee C \vee \neg E \vee F) \wedge (B \vee C \vee \neg E \vee F) \wedge (A \vee D \vee \neg E \vee F)$ $\wedge (B \vee D \vee \neg E \vee F)$.

13. **a.** Full DNF: $(P \wedge Q) \vee (P \wedge \neg Q)$. Full CNF: $(P \vee \neg Q) \wedge (P \vee Q)$.

14. **a.** $(P \wedge Q) \vee (P \wedge \neg Q)$.
c. $(P \wedge Q) \vee (P \wedge \neg Q) \vee (\neg P \wedge Q) \vee (\neg P \wedge \neg Q)$.
e. $(P \wedge Q \wedge R) \vee (\neg P \wedge Q \wedge R) \vee (\neg P \wedge \neg Q \wedge R)$ $\vee (\neg P \wedge Q \wedge \neg R) \vee (\neg P \wedge \neg Q \wedge \neg R)$.

15. **a.** $(P \vee Q) \wedge (P \vee \neg Q)$. **c.** $\neg Q \vee P$. **e.** $(P \vee Q \vee R) \wedge (P \vee Q \vee \neg R)$ $\wedge (P \vee \neg Q \vee R) \wedge (\neg P \vee Q \vee R) \wedge (\neg P \vee \neg Q \vee R)$.

16. **a.** $A \vee B \equiv \neg(\neg A \wedge \neg B)$. So $\{\neg, \wedge\}$ is complete because $\{\neg, \vee\}$ is complete.
c. $\neg \text{A} \equiv A \to \text{false}$. So $\{\text{false}, \to\}$ is a complete set because $\{\neg, \to\}$ is complete.
e. $\neg A \equiv \text{NOR}(A, A)$, and $A \vee B \equiv \neg \text{NOR}(A, B) = \text{NOR}(\text{NOR}(A, B), \text{NOR}(A, B))$. Therefore, NOR is complete because $\{\neg, \vee\}$ is a complete set of connectives.

Section 6.3

1. **a.** Line 2 is incorrect, since no part of W requires us to prove something of the form $A \to X$.

2. Line 6 is not correct because it uses line 3, which is in a previous subproof. Only lines 1 and 5 can be used to infer something on line 6.

3. **a.** Three premises: A is the premise for the proof of the conditional, whose conclusion is $B \to (C \to D)$. B is the premise for the conditional proof whose conclusion is $C \to D$. Finally, C is the premise for the proof of $C \to D$.

4. Let D mean "I am dancing," H mean "I am happy," and M mean "There is a mouse in the house." Then a proof can be written as follows:

1.	$D \to H$	P
2.	$M \vee H$	P
3.	$\neg H$	P
4.	M	2, 3, DS
5.	$\neg D$	1, 3, MT
6.	$M \wedge \neg D$	4, 5, Conj
	QED	1, 2, 3, 6, CP.

5. a.

1.	A	P
2.	$\qquad B$	P
3.	$\qquad A \wedge B$	1, 2, Conj
4.	$B \to A \wedge B$	2, 3, CP
	QED	1, 4, CP.

c.

1.	$A \vee B \to C$	P
2.	A	P
3.	$A \vee B$	2, Add
4.	C	1, 3, MP
	QED	1, 2, 4, CP.

e.

1.	$A \vee B \to C \wedge D$	P
2.	$\qquad B$	P
3.	$\qquad A \vee B$	2, Add
4.	$\qquad C \wedge D$	1, 3, MP
5.	$\qquad D$	4, Simp
6.	$B \to D$	2, 5, CP
	QED	1, 6, CP.

g.

1.	$\neg (A \wedge B)$	P
2.	$B \vee C$	P
3.	$C \to D$	P
4.	$\qquad A$	P
5.	$\qquad \neg A \vee \neg B$	1, T
6.	$\qquad \neg B$	4, 5, DS
7.	$\qquad C$	2, 6, DS
8.	$\qquad D$	3, 7, MP
9.	$A \to D$	4, 8, CP
	QED	1, 2, 3, 9, CP.

i.

1.	$A \to C$	P
2.	$\qquad A \wedge B$	P
3.	$\qquad A$	2, Simp
4.	$\qquad C$	1, 3, MP
5.	$A \wedge B \to C$	2, 4, CP
	QED	1, 5, CP.

6. a.

1.	A	P
2.	$\neg (B \rightarrow A)$	P for IP
3.	$\neg (\neg B \vee A)$	2, T
4.	$B \wedge \neg A$	3, T
5.	$\neg A$	4, Simp
6.	$A \wedge \neg A$	1, 5, Conj
	QED	1, 2, 6, IP.

c.

1.	$\neg B$	P
2.	$\neg (B \rightarrow C)$	P for IP
3.	$\neg (\neg B \vee C)$	2, T
4.	$B \wedge \neg C$	3, T
5.	B	4, Simp
6.	$\neg B \wedge B$	1, 5, Conj
	QED	1, 2, 6, IP.

e.

1.	$A \rightarrow B$	P
2.	$\neg ((A \rightarrow \neg B) \rightarrow \neg A)$	P for IP
3.	$(\neg A \vee \neg B) \wedge A$	2, T
4.	A	3, Simp
5.	$\neg A \vee \neg B$	3, Simp
6.	$\neg B$	4, 5, DS
7.	$\neg A$	1, 6, MT
8.	$A \wedge \neg A$	4, 7, Conj
	QED	1, 2, 8, IP.

g.

1.	$A \rightarrow B$	P
2.	$B \rightarrow C$	P
3.	$\neg (A \rightarrow C)$	P for IP
4.	$A \wedge \neg C$	3, T
5.	A	4, Simp
6.	$\neg C$	4, Simp
7.	B	1, 5, MP
8.	C	2, 7, MP
9.	$C \wedge \neg C$	6, 8, Conj
	QED	1, 2, 3, 9, IP.

7. For some proofs we'll use IP in a subproof.

a.

1.	A	P
2.	$\neg (B \to (A \wedge B))$	P for IP
3.	$B \wedge \neg (A \wedge B)$	2, T
4.	B	3, Simp
5.	$\neg (A \wedge B)$	3, Simp
6.	$\neg A \vee \neg B$	5, T
7.	$\neg B$	1, 6, DS
8.	$B \wedge \neg B$	4, 7, Conj
9.	false	8, T
	QED	1, 2, 9, IP.

c.

1.	$A \vee B \to C$	P
2.	A	P
3.	$\neg C$	P for IP
4.	$\neg (A \vee B)$	1, 3, MT
5.	$\neg A \wedge \neg B$	4, T
6.	$\neg A$	5, Simp
7.	$A \wedge \neg A$	2, 6, Conj
	QED	1, 2, 3, 7, IP.

e.

1.	$A \vee B \to C \wedge D$	P
2.	$\neg (B \to D)$	P for IP
3.	$B \wedge \neg D$	2, T
4.	B	3, Simp
5.	$A \vee B$	4, Add
6.	$C \wedge D$	1, 5, MP
7.	$\neg D$	3, Simp
8.	D	6, Simp
9.	$D \wedge \neg D$	7, 8, Conj
	QED	1, 2, 9, IP.

g.

1.	$\neg (A \wedge B)$	P
2.	$B \vee C$	P
3.	$C \rightarrow D$	P
4.	$\neg (A \rightarrow D)$	P for IP
5.	$A \wedge \neg D$	4, T
6.	$\neg D$	5, Simp
7.	$\neg C$	3, 6, MT
8.	B	2, 7, DS
9.	$B \rightarrow \neg A$	1, T
10.	$\neg A$	8, 9, MP
11.	A	5, Simp
12.	$A \wedge \neg A$	10, 11, Conj
	QED	1, 2, 3, 4, 12, IP.

i.

1.	$A \rightarrow (B \rightarrow C)$	P
2.	$\quad B$	P
3.	$\quad\quad A$	P
4.	$\quad\quad \neg C$	P for IP
5.	$\quad\quad B \rightarrow C$	1, 3, MP
6.	$\quad\quad C$	2, 5, MP
7.	$\quad\quad C \wedge \neg C$	4, 6, Conj
8.	$\quad A \rightarrow C$	3, 4, 7, IP
9.	$B \rightarrow (A \rightarrow C)$	2, 8, CP
	QED	1, 9, CP.

k.

1.	$A \rightarrow C$	P
2.	$\quad A$	P
3.	$\quad \neg (B \vee C)$	P for IP
4.	$\quad \neg B \wedge \neg C$	3, T
5.	$\quad C$	1, 2, MP
6.	$\quad \neg C$	4, Simp
7.	$\quad C \wedge \neg C$	5, 6, Conj
8.	$A \rightarrow B \vee C$	2, 3, 7, IP
	QED	1, 8, CP.

8. a.

1.	$(A \wedge B) \rightarrow C$			P
2.		A		P
3.			B	P
4.			$A \wedge B$	2, 3, Conj
5.			C	1, 4, MP
6.		$B \rightarrow C$		3, 5, CP
7.	$A \rightarrow (B \rightarrow C)$			2, 6, CP
	QED			1, 7, CP.

9. a.

1.	$A \vee B$	P
2.	$A \rightarrow C$	P
3.	$B \rightarrow D$	P
4.	$\neg (C \vee D)$	P for IP
5.	$\neg C \wedge \neg D$	4, T
6.	$\neg C$	5, Simp
7.	$\neg A$	2, 6, MT
8.	B	1, 7, DS
9.	D	3, 8, MP
10.	$\neg D$	5, Simp
11.	$D \wedge \neg D$	9, 10, Conj
	QED	1, 2, 3, 4, 11, IP.

10. a.

1.	$A \rightarrow B$		P
2.	$\neg A \rightarrow C$		P
3.		A	P
4.		B	1, 3, MP
5.		$A \wedge B$	3, 4, Conj
6.	$A \rightarrow A \wedge B$		3, 5, CP
7.		$\neg A$	P
8.		C	2, 7, MP
9.		$\neg A \wedge C$	7, 8, Conj
10.	$\neg A \rightarrow \neg A \wedge C$		7, 9, CP
11.	$A \wedge \neg A$		T
12.	$(A \wedge B) \vee (\neg A \wedge C)$		6, 10, 11, CD
	QED		1, 2, 12, CP.

Section 6.4

1.

1.	$A \to (B \to C)$	Premise
2.	$(A \to (B \to C)) \to ((A \to B) \to (A \to C))$	Axiom 2
3.	$(A \to B) \to (A \to C)$	1, 2, MP
4.	B	Premise
5.	$B \to (A \to B)$	Axiom 1
6.	$A \to B$	4, 5, MP
7.	$A \to C$	3, 6, MP
8.	$B \to (A \to C)$	4, 7, CP
	QED	1, 8, CP.

3. a.

1.	$(A \to B) \to ((\neg C \vee A) \to (\neg C \vee B))$	Axiom 4
2.	$(A \to B) \to ((C \to A) \to (C \to B))$	1, Definition of $\to$
	QED.	

c.

1.	$A \to A \vee A$	Axiom 2
2.	$A \vee A \to A$	Axiom 1
3.	$A \to A$	1, 2, HS (i.e., Part (b))
	QED.	

e.

1.	$\neg A \vee \neg \neg A$	Part (d)
2.	$A \to \neg \neg A$	1, Definition of $\to$
	QED.	

g.

1.	$\neg A \to (\neg A \vee B)$	Axiom 2
2.	$\neg A \to (A \to B)$	1, definition of $\to$
	QED.	

i.

1.	$\neg \neg B \to B$	Part (e)
2.	$(\neg \neg B \to B) \to ((\neg A \vee \neg \neg B) \to (\neg A \vee B))$	Axiom 4
3.	$(\neg A \vee \neg \neg B) \to (\neg A \vee B)$	1, 2, MP
4.	$(\neg \neg B \vee \neg A) \to (\neg A \vee \neg \neg B)$	Axiom 3
5.	$(\neg \neg B \vee \neg A) \to (\neg A \vee B)$	3, 4, HS (i.e., Part (b))
6.	$(\neg B \to \neg A) \to (A \to B)$	5, Definition of $\to$
	QED.	

5. We give a couple hints to aid the reasoning process. *Hint:* Let each name, like A, mean that A won a position. Then transform each statement into a wff of the propositional calculus. Create a wff to describe the problem, and find an assignment of truth values to make the wff true. *Hint:* Make a table of possibilities with rows A, B, C, and D, and columns E, F, G, and H. Place a check in an entry if the row name and column name were not elected to the board.

Chapter 7

Section 7.1

1. a. $[p(0, 0) \wedge p(0, 1)] \vee [p(1, 0) \wedge p(1, 1)]$.

2. a. $\forall x\ q(x)$, where $x \in \{0, 1\}$.
c. $\forall\ y\ p(x, y)$, where $y \in \{0, 1\}$.
e. $\exists x\ p(x)$, where x is an odd natural number.

3. a. x is a term. Therefore, $p(x)$ is a wff, and it follows that $\exists x\ p(x)$ and $\forall x\ p(x)$ are wffs. Thus $\exists x\ p(x) \to \forall x\ p(x)$ is a wff.

4. It is illegal to have an atom, $p(x)$ in this case, as an argument to a predicate.

5. a. The three occurrences of x, left to right, are free, bound, and bound. The four occurrences of y, left to right, are free, bound, bound, and free.
c. The three occurrences of x, left to right, are free, bound, and bound. Both occurrences of y are free.

6. $\forall x\ p(x, y, z) \to \exists\ z\ q(z)$.

7. a. Every bird eats every worm.

8. a. There is someone who eats every chocolate bar.

9. a. $\forall x\ (\neg\ e(x, a) \to \exists y\ p(y, x))$, where $a = 0$.

10. a. One interpretation has $p(a)$ = true, in which case both $\forall x\ p(x)$ and $\exists x\ p(x)$ are true. Therefore, W is true. The other interpretation has $p(a)$ = false, in which case both $\forall x\ p(x)$ and $\exists x\ p(x)$ are false. Therefore, W is true.

11. a. Let the domain be the set $\{a, b\}$, and assign $p(a)$ = true and $p(b)$ = false. Finally, assign the constant $c = a$.
c and **d.** Let $p(x, y)$ = false for all elements x and y in any domain. Then the antecedent is false for both parts (c) and (d). Therefore, both wffs are true for this interpretation. **e.** Let $D = \{a\}$, $f(a) = a$, $y = a$, and let p denote equality.

12. a. Let the domain be $\{a\}$, and let $p(a)$ = true and $c = a$.
c. Let $D = \mathbb{N}$, let $p(x)$ mean "x is odd," and let $q(x)$ mean "x is even." Then the antecedent is true, but the consequent is false.
e. Let $D = \mathbb{N}$, and let $p(x, y)$ mean "$y = x + 1$." Then the antecedent $\forall x\ \exists y\ p(x, y)$ is true and the consequent $\exists y\ \forall x\ p(x, y)$ is false for this interpretation.
g. Let $D = \{a, b\}$, $p(a)$ = true, $p(b)$ = false, $q(a)$ = false, and $q(b)$ = true. Then $\forall x\ p(x)$ is false, so the antecedent is true. But $p(a) \to q(a)$ is false, so the consequent is false.

13. a. If the domain is $\{a\}$, then either $p(a)$ = true or $p(a)$ = false. In either case, W is true.

14. a. Let $\{a\}$ be the domain of the interpretation. If $p(a, a)$ = false, then W is true, since the antecedent is false. If $p(a, a)$ = true, the W is true, since the consequent is true.
c. Let $\{a, b, c\}$ be the domain. Let $p(a, a) = p(b, b) = p(c, c)$ = true and $p(a, b) = p(a, c) = p(b, c)$ = false. This assignment makes W false. Therefore, W is invalid.

15. $\forall x\ p(x, x) \rightarrow$
$\forall x\ \forall y\ \forall z\ \forall w\ (p(x, y) \vee p(x, z) \vee p(x, w) \vee p(y, z) \vee p(y, w) \vee p(z, w))$.

16. a. For any domain D and any element $d \in D$, $p(d) \rightarrow p(d)$ is true. Therefore, any interpretation is a model.

c. If the wff is invalid, then there is some interpretation making the wff false. This says that $\forall x\, p(x)$ is true and $\exists x\, p(x)$ is false. This is a contradiction because we can't have $p(x)$ true for all x in a domain while at the same time having $p(x)$ false for some x in the domain.

e. If the wff is not valid, then there is an interpretation with domain D for which the antecedent is true and the consequent is false. So $A(d)$ and $B(d)$ are false for some element $d \in D$. Therefore, $\forall x\ A(x)$ and $\forall x\ B(x)$ are false, contrary to assumption.

g and **h.** If the antecedent is true for a domain D, then $A(d) \rightarrow B(d)$ is true for all $d \in D$. If $A(d)$ is true for all $d \in D$, then $B(d)$ is also true for all $d \in D$ by MP. Thus the consequent is true for D.

17. a. Suppose the wff is satisfiable. Then there is an interpretation that assigns c a value in its domain such that $p(c) \wedge \neg\ p(c) =$ true. Of course, this is impossible. Therefore, the wff is unsatisfiable.

c. Suppose the wff is satisfiable. Then there is an interpretation making $\exists x\ \forall y\ (p(x, y) \wedge \neg\ p(x, y))$ true. This says that there is an element d in the domain such that $\forall y\ (p(d, y) \wedge \neg\ p(d, y))$ is true. This says that $p(d, y) \wedge \neg\ p(d, y)$ is true for all y in the domain, which is impossible.

19. Assume that $A \rightarrow B$ is valid and A is also valid. Let I be an interpretation for B with domain D. Extend I to an interpretation J for A by using D to interpret all predicates, functions, free variables and constants that occur in A but not in B. So J is an interpretation for $A \rightarrow B$, A, and B. Since we are assuming that $A \rightarrow B$ and A are valid, it follows that $A \rightarrow B$ and A are true with respect to J. Therefore, B is true with respect to J. But J and I are the same interpretation on B. So B is true with respect to I. Therefore, I is a model for B. Since I was arbitrary, it follows that B is valid. Now we go the other direction. Assume that if A is valid, then B is valid. Let I be an interpretation for $A \rightarrow B$. Then I is also an interpretation for A and for B. Since A and B are valid, it follows that A and B are true with respect to I. Therefore, $A \rightarrow B$ is true with respect to I. Therefore, I is a model for $A \rightarrow B$. Since I was arbitrary, it follows that $A \rightarrow B$ is valid.

Section 7.2

1. a. The left side is true for domain D iff $A(d) \wedge B(d)$ is true for all $d \in D$ iff $A(d)$ and $B(d)$ are both true for all $d \in D$ iff the right side is true for D.

c. Assume that the left side is true for domain D. Then $A(d) \rightarrow B(d)$ is true for some $d \in D$. If $A(d)$ is true, then $B(d)$ is true by MP. So $\exists x\ B(x)$ is true for D. If $A(d)$ is false, then $\forall\ x\ A(x)$ is false. So in either case the right side is true for D. Now assume the right side is true for D. If $\forall x\ A(x)$ is true, then $\exists x\ B(x)$ is

also true. This means that $A(d)$ is true for all $d \in D$ and $B(d)$ is true for some $d \in D$. Thus $A(d) \rightarrow B(d)$ is true for some $d \in D$, which says that the left side is true for D. If $\forall x\ A(x)$ is false, then $A(d)$ is false for some $d \in D$. So $A(d) \rightarrow B(d)$ is true. Thus the left side is true for D.
e. $\exists x\ \exists y\ W(x, y)$ is true for D iff $W(d, e)$ is true for some elements $d, e \in D$ iff $\exists y\ \exists\ x\ W(x, y)$ is true for D.

2. The assumption that x is not free in C means that any substitution x/t does not change C. In other words, $C(x/t) = C$ for all possible terms t. We'll assume that I is an interpretation with domain D.
a. If I is a model for $\forall x\ C$, then $C(x/d)$ is true for I for all d in D. Since $C(x/d) = C$, it follows that C is true for I. Therefore, I is a model for C. If I is a model for C, then C is true for I. Since $C = C(x/d)$ for all d in D, it follows that $C(x/d)$ is true for I for all d in D. Therefore, I is a model for $\forall\ x\ C$.
c. If I is a model for $\exists x\ (C \vee A(x))$, then $(C \vee A(x))(x/d)$ is true for I for some d in D. But we have $(C \vee A(x))(x/d) = C(x/d) \vee A(x)(x/d) = C \vee A(x)(x/d)$ because x is not free in C. So $C \vee A(x)(x/d)$ is true for I. Since C is not affected by any substitution for x, it follows that either C is true for I or $A(x)(x/d)$ is true for I. So either I is a model for C or I is a model for $\exists x\ A(x)$. Therefore, I is a model for $C \vee \exists x\ A(x)$.

Conversely, if I is a model for $C \vee \exists x\ A(x)$, then $C \vee \exists x\ A(x)$ is true for I. So either C is true for I or $\exists x\ A(x)$ is true for I. Suppose C is true for I. Since $C = C(x/d)$ for any d in D, it is true for some d in D. So $C(x/d) \vee A(x)(x/d)$ is true for I for some d in D. Subtitution gives $C(x/d) \vee A(x)(x/d) = (C \vee A(x))(x/d)$. So $(C \vee A(x))(x/d)$ is true for I for some d in D. Thus I is a model for $\exists\ x\ (C \vee A(x))$. Suppose that $\exists x\ A(x)$ is true for I, then $A(x)(x/d)$ is true for I for some d in D. So $C(x/d) \vee A(x)(x/d)$ is true for I and thus $(C \vee A(x))(x/d)$ is true for I. So I is a model for $\exists x\ (C \vee A(x))$.

3. a. $\forall x\ (C \rightarrow A(x)) \equiv \forall x\ (\neg\ C \vee A(x)) \equiv \neg\ C \vee \forall\ x\ A(x) \equiv C \rightarrow \forall x\ A(x)$.
c. $\exists x\ (A(x) \rightarrow C) \equiv \exists x\ (\neg\ A(x) \vee C) \equiv \exists\ x\ \neg\ A(x) \vee C \equiv \neg\ \forall\ x\ A(x) \vee C \equiv \forall x\ A(x) \rightarrow C$.

4. a. $\exists x\ \forall\ y\ \forall z\ ((\neg\ p(x) \vee p(y) \vee q(z)) \wedge (\neg\ q(x) \vee p(y) \vee q(z)))$.
c. $\exists x\ \forall y\ \exists z\ \forall w\ (\neg\ p(x, y) \vee p(w, z))$.
e. $\forall x\ \forall y\ \exists z\ ((\neg\ p(x, y) \vee p(x, z)) \wedge (\neg\ p(x, y) \vee p(y, z)))$.

5. a. $\exists x\ \forall\ y\ \forall z\ ((\neg\ p(x) \wedge \neg\ q(x)) \vee p(y) \vee q(z))$.
c. $\exists x\ \forall y\ \exists z\ \forall w\ (\neg\ p(x, y) \vee p(w, z))$.
e. $\forall x\ \forall y\ \exists z\ (\neg\ p(x, y) \vee (p(x, z) \wedge p(y, z)))$.

6. a. Let D be the domain $\{a, b\}$. Assume that C is false, $W(a)$ is true, and $W(b)$ is false. Then $(\forall x\ W(x) \equiv C)$ is true, but $\forall x\ (W(x) \equiv C)$ is false. Therefore, the statement is false.

7. a. $\forall x\ (C(x) \rightarrow R(x) \wedge F(x))$.
c. $\forall x\ (G(x) \rightarrow S(x))$.
e. $\exists x\ (G(x) \wedge \neg\ S(x))$.

8. a. $\forall x\ (B(x) \rightarrow \exists y\ (W(y) \wedge E(x, y)))$.

c. $\forall x\, \forall y\, (W(x) \wedge E(y, x) \rightarrow B(y))$.
e. $\forall x\, \forall y\, (B(x) \wedge E(x, y) \rightarrow W(y))$.
g. $\exists x\, (\neg\, B(x) \wedge \exists y\, (W(y) \wedge E(x, y)))$.

9. a. $\forall x\, (F(x) \rightarrow S(\text{x})) \wedge \neg\, S(\text{John}) \rightarrow \neg\, F(\text{John})$.

10. a. $\forall x\, (P(x) \wedge \neg\, B(x) \rightarrow \forall y\, (C(y) \rightarrow K(x, y)))$.
c. $\forall x\, (B(x) \rightarrow \neg\, K(x, x))$.
e. $\forall x\, (P(x) \rightarrow \exists y\, (P(y) \wedge N(x, y) \wedge G(y)))$.
g. $\forall x\, \forall y\, (P(x) \wedge A(y) \wedge \neg\, K(x, y) \rightarrow \neg\, G(x))$.

Section 7.3

1. a. Line 2 is wrong because x is free in line 1, which is a premise.So line 1 can't be used with the UG rule to generalize x.
c. Line 2 is wrong because $f(y)$ is not free to replace x. That is, the substitution of $f(y)$ for x yields a new bound occurrence of y. Therefore, EG can't generalize to x from $f(y)$.
e. Line 4 is wrong because c already occurs in the proof on line 3.

2. Lines 3 and 4 are errors; they apply UG to a subexpression of a larger wff.

3. a. Let $D = \mathbb{N}$, let $P(x) =$ "$x + x = x$," and $Q(x) =$ "$x + 1 = x$." Then $P(0)$ is true, and $P(1) \rightarrow Q(1)$ is also true. Therefore, the antecedent of the wff is true. But $Q(x)$ is false for all x in $\mathbb{N}$. Therefore, the consequent is false. Therefore, the interpretation is a countermodel, and the wff is invalid.

5. This reasoning is wrong because it assumes that CP is an inference rule. But CP is a proof rule, not an inference rule. So UG can be applied to line 3.

6. a.

1.	$\forall x\, p(x)$	P
2.	$p(x)$	1, UI
3.	$\exists x\, p(x)$	2, EG
	QED	1, 3, CP.

c.

1.	$\exists x\, (p(x) \wedge q(x))$	P
2.	$p(c) \wedge q(c)$	1, EI
3.	$p(c)$	2, Simp
4.	$\exists x\, p(x)$	3, EG
5.	$q(c)$	2, Simp
6.	$\exists x\, q(x)$	5, EG
7.	$\exists x\, p(x) \wedge \exists x\, q(x)$	4, 6, Conj
	QED	1, 7, CP.

e.

1.	$\forall x\ (p(x) \to q(x))$	P
2.	$\forall x\ p(x)$	P
3.	$p(x)$	2, UI
4.	$p(x) \to q(x)$	1, UI
5.	$q(x)$	3, 4, MP
6.	$\exists x\ q(x)$	5, EG
7.	$\forall x\ p(x) \to \exists x\ q(x)$	2, 6, CP
	QED	1, 7, CP.

g.

1.	$\exists y\ \forall x\ p(x, y)$	P
2.	$\forall x\ p(x, c)$	1, EI
3.	$p(x, c)$	2, UI
4.	$\exists y\ p(x, y)$	3, EG
5.	$\forall x\ \exists y\ p(x, y)$	4, UG
	QED	1, 5, CP.

7. a.

1.	$\forall x\ p(x)$	P
2.	$\neg\ \exists x\ p(x)$	P for IP
3.	$\forall x\ \neg\ p(x)$	2, T
4.	$p(x)$	1, UI
5.	$\neg\ p(x)$	3, UI
6.	$p(x) \wedge \neg\ p(x)$	4, 5, Conj
7.	false	6, T
	QED	1, 2, 7, IP.

c.

1.	$\exists y\ \forall x\ p(x, y)$	P
2.	$\neg\ \forall x\ \exists y\ p(x, y)$	P for IP
3.	$\exists x\ \forall y\ \neg\ p(x, y)$	2, T
4.	$\forall x\ p(x, c)$	1, EI
5.	$\forall y\ \neg\ p(d, y)$	3, EI
6.	$p(d, c)$	4, UI
7.	$\neg\ p(d, c)$	5, UI
8.	$p(d, c) \wedge \neg\ p(d, c)$	6, 7, Conj
	QED	1, 2, 8, IP.

e.

1.	$\forall x\, p(x) \vee \forall x\, q(x)$	P
2.	$\neg \forall x\, (p(x) \vee q(x))$	P for IP
3.	$\exists x\, (\neg p(x) \wedge \neg q(x))$	2, T
4.	$\neg p(c) \wedge \neg q(c)$	3, EI
5.	$\neg p(c)$	4, Simp
6.	$\neg \forall x\, p(x)$	5, UI (contrapositive)
7.	$\neg q(c)$	4, Simp
8.	$\neg \forall x\, q(x)$	7, UI (contrapositive)
9.	$\neg \forall x\, p(x) \wedge \neg \forall x\, q(x)$	6, 7, Conj
10.	$\neg (\forall x\, p(x) \vee \forall x\, q(x))$	9, T
11.	false	1, 10, Conj, T
	QED	1, 2, 11, IP.

8. a. Let $D(x)$ mean that x is a dog, $L(x)$ mean that x likes people, $H(x)$ mean that x hates cats, and a = Rover. Then the argument can be formalized as

$$\forall x\, (D(x) \to L(x) \vee H(x)) \wedge D(a) \wedge \neg H(a) \to \exists x\, (D(x) \wedge L(x)).$$

Proof:	1.	$\forall x\, (D(x) \to L(x) \vee H(x))$	P
	2.	$D(a)$	P
	3.	$\neg H(a)$	P
	4.	$D(a) \to L(a) \vee H(a)$	1, UI
	5.	$L(a) \vee H(a)$	2, 4, MP
	6.	$L(a)$	3, 5, DS
	7.	$D(a) \wedge L(a)$	2, 6, Conj
	8.	$\exists x\, (D(x) \wedge L(x))$	7, EG
		QED	1, 2, 3, 8, CP.

c. Let $H(x)$ mean that x is a human being, $Q(x)$ mean that x is a quadruped, and $M(x)$ mean that x is a man. Then the argument can be formalized as

$$\forall x\, (H(x) \to \neg Q(x)) \wedge \forall x\, (M(x) \to H(x)) \to \forall x\, (M(x) \to \neg Q(x)).$$

Proof:	1.	$\forall x\, (H(x) \to \neg Q(x))$	P
	2.	$\forall x\, (M(x) \to H(x))$	P
	3.	$H(x) \to \neg Q(x)$	1, UI
	4.	$M(x) \to H(x)$	2, UI
	5.	$M(x) \to \neg Q(x)$	3, 4, HS
	6.	$\forall x\, (M(x) \to \neg Q(x))$	5, UG
		QED	1, 2, 6, CP.

e. Let $F(x)$ mean that x is a freshman, $S(x)$ mean that x is a sophomore, $J(x)$ mean that x is a junior, and $L(x, y)$ mean that x likes y. Then the argument can be formalized as $A \to B$, where

$$A = \exists x\,(F(x) \wedge \forall y\,(S(y) \to L(x,y))) \wedge \forall x\,(F(x) \to \forall y\,(J(y) \to \neg\, L(x,y)))$$
$$B = \forall x\,(S(x) \to \neg\, J(x)).$$

Proof:

1.	$\exists x\ (F(x) \wedge \forall y\ (S(y) \to L(x, y)))$	P
2.	$\forall x\ (F(x) \to \forall y\ (J(y) \to \neg\, L(x, y)))$	P
3.	$F(c) \wedge \forall y\ (S(y) \to L(c, y))$	1, EI
4.	$\forall y\ (S(y) \to L(c, y))$	3, Simp
5.	$S(x) \to L(c, x)$	4, UI
6.	$\quad S(x)$	P
7.	$\quad L(c, x)$	5, 6, MP
8.	$\quad F(c) \to \forall y\ (J(y) \to \neg\, L(c, y))$	2, UI
9.	$\quad F(c)$	3, Simp
10.	$\quad \forall y\ (J(y) \to \neg\, L(c, y))$	8, 9, MP
11.	$\quad J(x) \to \neg\, L(c, x)$	10, UI
12.	$\quad \neg\, J(x)$	7, 11, MT
13.	$S(x) \to \neg\, J(x)$	6, 12, CP
14.	$\forall x\ (S(x) \to \neg\, J(x))$	13, UG
	QED	1, 2, 14, CP.

9. First prove that the left side implies the right side, then the converse.

a.

1.	$\exists x\, \exists y\ W(x, y)$	P
2.	$\exists y\ W(c, y)$	1, EI
3.	$W(c, d)$	2, EI
4.	$\exists x\ W(x, d)$	3, EG
5.	$\exists y\, \exists x\ W(x, y)$	4, EG
	QED	1, 5, CP.

1.	$\exists y\, \exists x\ W(x, y)$	P
2.	$\exists x\ W(x, d)$	1, EI
3.	$W(c, d)$	2, EI
4.	$\exists y\ W(c, y)$	3, EG
5.	$\exists x\, \exists y\ W(x, y)$	4, EG
	QED	1, 5, CP.

c.	1.	$\exists x\ (A(x) \lor B(x))$	P
	2.	$\neg\ (\exists x\ A(\mathrm{x}) \lor \exists x\ B(x))$	P for IP
	3.	$\forall x \neg\ A(x) \land \forall x \neg\ B(x)$	2, T
	4.	$\forall x \neg\ A(x)$	3, Simp
	5.	$A(c) \lor B(c)$	1, EI
	6.	$\neg\ A(c)$	4, UI
	7.	$B(c)$	5, 6, DS
	8.	$\forall x \neg\ B(x)$	3, Simp
	9.	$\neg\ B(c)$	8, UI
	10.	$B(c) \land \neg\ B(c)$	7, 9, Conj
		QED	1, 2, 10, IP.

	1.	$\exists x\ A(x) \lor \exists x\ B(x)$	P
	2.	$\neg \exists x\ (A(x) \lor B(x))$	P for IP
	3.	$\forall x\ (\neg\ A(x) \land \neg\ B(x))$	2, T
	4.	$\forall\ x \neg\ A(x) \land \forall x \neg\ B(x)$	3, T Part (b)
	5.	$\forall x \neg\ A(x)$	4, Simp
	6.	$\neg \exists x\ A(x)$	5, T
	7.	$\exists x\ B(x)$	1, 6, DS
	8.	$\forall x \neg\ B(x)$	4, Simp
	9.	$\neg \exists x\ B(\mathrm{x})$	5, T
	10.	$\exists x\ B(x) \land \neg \exists x\ B(x)$	7, 9, Conj
		QED	1, 2, 10, IP.

10.	1.	$\forall x\ (\exists y\ (q(x,\ y) \land s(y)) \to \exists y\ (p(y) \land r(x,\ y)))$	P
	2.	$\neg\ (\neg \exists x\ p(x) \to \forall x\ \forall y\ (q(x,\ y) \to \neg\ s(y)))$	P for IP
	3.	$\neg \exists x\ p(x) \land \neg \forall\ x\ \forall y\ (q(x,\ y) \to \neg\ s(y))$	2, T
	4.	$\neg \exists x\ p(x)$	3, Simp
	5.	$\neg \forall x\ \forall y\ (q(x,\ y) \to \neg\ s(y))$	3, Simp
	6.	$\exists x\ \exists y\ (q(x,\ y) \land s(y))$	5, T
	7.	$\exists y\ (q(c,\ y) \land s(y))$	6, EI
	8.	$\exists y\ (q(c,\ y) \land s(y)) \to \exists y\ (p(y) \land r(c,\ y))$	1, UI
	9.	$\exists y\ (p(y) \land r(c,\ y))$	7, 8, MP
	10.	$p(d) \land r(c,\ d)$	9, EI
	11.	$p(d)$	10, Simp
	12.	$\exists x\ p(x)$	11, EG
	13.	false	4, 12, Conj, T
		QED	1, 2, 13, IP.

12. a.

1.	$\exists\, x\, A(x)$	P
2.	$A(x)$	1, EI (wrong to use a variable)
3.	$\forall x\, A(x)$	2, UG.

c.

1.	$\forall x\, (A(x) \vee B(x))$	P
2.	$\neg\, (\forall x\, A(x) \vee \forall x\, B(x))$	P for IP
3.	$\exists x \neg\, A(x) \wedge \exists x \neg\, B(x)$	2, T
4.	$\exists x \neg\, A(x)$	3, Simp
5.	$\exists x \neg\, B(x)$	4, Simp
6.	$\neg\, A(c)$	4, EI
7.	$\neg\, B(c)$	5, EI (wrong to use an existing constant)
8.	$A(c) \vee B(c)$	1, UI
9.	$B(c)$	6, 8, DS
10.	false	7, 9, Conj, T.

e.

1.	$\forall x\, \exists y\, W(x, y)$	P
2.	$\exists y\, W(x, y)$	1, UI
3.	$W(x, c)$	2, EI
4.	$\forall x\, W(x, c)$	3, UG (wrong because x is subscripted)
5.	$\exists y\, \forall x\, W(x, y)$	4, EG (may be wrong if $W(x, c)$ contains bound y).

13. a. Similar to proof of Exercise 9b.
c. Use IP in both directions.
e. Similar to part (c). **g.** Similar to part (c).

14. a. Let $\neg\, B$ and $A \rightarrow B$ be valid wffs. Consider an arbitrary interpretation of these two wffs with domain D. Then $\neg\, B$ and $A \rightarrow B$ are true for D. Thus we can apply MT to conclude that $\neg\, A$ is true for D. Since the interpretation was arbitrary, it follows that $\neg\, A$ is valid.

15. a. The variable x is not free within the scope of a quantifier in $W(x)$.

17. Proof:

1.	$\forall x \neg p(x, x)$	P
2.	$\forall x \forall y \forall z\ (p(x, y) \wedge p(y, z) \rightarrow p(x, z))$	P
3.	$\neg \forall x \forall y\ (p(x, y) \rightarrow \neg p(y, x))$	P for IP
4.	$\exists x \exists y\ (p(x, y) \wedge p(y, x))$	3, T
5.	$\exists y\ (p(a, y) \wedge p(y, a))$	4, EI
6.	$p(a, b) \wedge p(b, a))$	5, EI
7.	$\forall y \forall z\ (p(a, y) \wedge p(y, z) \rightarrow p(a, z))$	2, UI
8.	$\forall z\ (p(a, b) \wedge p(b, z) \rightarrow p(a, z))$	7, UI
9.	$p(a, b) \wedge p(b, a) \rightarrow p(a, a))$	8, UI
10.	$p(a, a)$	6, 9, MP
11.	$\neg p(a, a)$	1, UI
12.	$p(a, a) \wedge \neg p\ (a, a)$	10, 11, Conj
13.	false	12, T
	QED	1, 2, 3, 12, IP.

Chapter 8

Section 8.1

1.

1.	$s = v$	P
2.	$t = w$	P
3.	$p(s, t)$	P
4.	$p(v, t)$	1, 3, EE
5.	$p(v, w)$	2, 4, EE
	QED	1, 2, 3, 5, CP.

3.

1.	$c = a^i$	P
2.	$i \leq b$	P
3.	$\neg (i < b)$	P
4.	$(i < b) \vee (i = b)$	2, T
5.	$i = b$	3, 4, DS
6.	$c = a^b$	1, 5, EE
	QED	1, 2, 3, 6, CP.

4. a.

1.	$t = u$	P
2.	$\neg p(\ldots t \ldots)$	P
3.	$p(\ldots u \ldots)$	P for IP
4.	$u = t$	1, Symmetric
5.	$p(\ldots t \ldots)$	3, 4, EE
6.	false	2, 5, Conj, T
	QED	1, 2, 3, 6, IP.

c.

1.	$t = u$	P
2.	$p(\ldots\ t\ \ldots) \vee q(\ldots\ t\ \ldots)$	P
3.	$\neg\ (p(\ldots\ u\ \ldots) \vee q(\ldots\ u\ \ldots))$	P for IP
4.	$\neg\ p(\ldots\ u\ \ldots) \wedge \neg\ q(\ldots\ u\ \ldots)$	3, T
5.	$\neg\ p(\ldots\ u\ \ldots)$	4, Simp
6.	$\neg\ q(\ldots\ u\ \ldots)$	4, Simp
7.	$u = t$	1, Symmetric
8.	$\neg\ p(\ldots\ t\ \ldots)$	5, 7, EE from part (a)
9.	$\neg\ q(\ldots\ t\ \ldots)$	6, 7, EE from part (a)
10.	$\neg\ p(\ldots\ t\ \ldots) \wedge \neg\ q(\ldots\ t\ \ldots)$	8, 9, Conj
11.	$\neg\ (p(\ldots\ t\ \ldots) \vee q(\ldots\ t\ \ldots))$	10, T
12.	false	2, 11, Conj, T
	QED	1, 2, 3, 12, IP.

e.

1.	$x = y$	P
2.	$\forall z\ p(\ldots\ x\ \ldots)$	P
3.	$p(\ldots\ x\ \ldots)$	2, UI
4.	$p(\ldots\ y\ \ldots)$	1, 3, EE
5.	$\forall z\ p(\ldots\ y\ \ldots)$	4, UG
	QED	1, 2, 5, CP.

5.

1.	$\neg\ \forall x\ \exists y\ (x = y)$	P for IP
2.	$\exists x\ \forall y\ (x \neq y)$	1, T
3.	$\forall y\ (c \neq y)$	2, EI
4.	$c \neq c$	3, UI
5.	$c = c$	EA
6.	false	4, 5, Conj, T
	QED	1, 6, IP.

6. a. Proof of $p(x) \rightarrow \exists y\ ((x = y) \wedge p(y))$:

1.	$p(x)$	P
2.	$\neg\ \exists\ y\ ((x = y) \wedge p(y))$	P for IP
3.	$\forall y\ ((x \neq y) \vee \neg\ p(y))$	2, T
4.	$(x \neq x) \vee \neg\ p(x)$	3, UI
5.	$x \neq x$	1, 4, DS
6.	$x = x$	EA
7.	false	5, 6, Conj, T
	QED	1, 2, 7, IP.

Proof of $\exists y\ ((x = y) \wedge p(y)) \to p(x)$:

1.	$\exists y\ ((x = y) \wedge p(y))$	P
2.	$(x = c) \wedge p(c)$	1, EI
3.	$p(x)$	2, EE
	QED	1, 3, CP.

7. a. $\text{odd}(x) = \exists z\ (x = 2z + 1)$.
c. $\text{div}(a, b) = (a \neq 0) \wedge \exists x\ (b = ax)$.
e. $\text{div}(d, a) \wedge \text{div}(d, b) \wedge \forall z\ (\text{div}(z, a) \wedge \text{div}(z, b) \to (z \leq d))$.

8. a. Possible answers include either of the following two equivalent wffs.

$$\forall x\ \forall y\ (A(x) \wedge A(y) \to (x = y)),$$
$$\neg\ \exists x\ A(x) \vee \exists x\ (A(x) \wedge \forall y\ (A(y) \to (x = y))).$$

c. One possible answer is $\forall x\ \forall y\ \forall z\ (A(x) \wedge A(y) \wedge A(z) \to (x = y) \vee (x = z) \vee (y = z))$. Another answer has the form: None $\vee$ Exactly One $\vee$ Exactly Two.

9. a. Proof that (a) implies (b).

1.	$\exists x\ (A(x) \wedge \forall y\ (A(y) \to (x = y)))$	P
2.	$\forall x\ \neg\ A(x) \vee \exists x\ \exists y\ (A(x) \wedge A(y) \wedge (x \neq y))$	P for IP
3.	$A(c) \wedge \forall y\ (A(y) \to (c = y))$	1, EI
4.	$A(c)$	3, Simp
5.	$\exists x\ A(x)$	5, EG
6.	$\neg\ \forall x\ \neg\ A(x)$	5, T
7.	$\exists x\ \exists\ y\ (A(x) \wedge A(y) \wedge (x \neq y))$	2, 6, DS
8.	$A(a) \wedge A(b) \wedge (a \neq b)$	7, EI, EI
9.	$\forall y\ (A(y) \to (c = y))$	3, Simp
10.	$A(a) \to (c = a)$	9, UI
11.	$A(a)$	8, Simp
12.	$c = a$	10, 11, MP
13.	$A(b) \to (c = b)$	9, UI
14.	$A(b)$	8, Simp
15.	$c = b$	13, 14, MP
16.	$a = b$	12, Symmetry, 15, Transitive
17.	$a \neq b$	8, Simp
18.	false	16, 17, Conj, T
	QED	1, 2, 18, IP.

Section 8.2

1.	1.	$\{\text{odd}(x + 1)\}\ y := x + 1\ \{\text{odd}(y)\}$	AA
	2.	$\quad\text{true} \wedge \text{even}(x)$	P
	3.	$\quad\text{even}(x)$	2, Simp
	4.	$\quad\text{odd}(x + 1)$	3, T
	5.	$\text{true} \wedge \text{even}(x) \to \text{odd}(x + 1)$	2, 4, CP
	6.	$\{\text{true} \wedge \text{even}(x)\}\ y := x + 1\ \{\text{odd}(y)\}$	1, 5, Consequence
		QED.	

2. a.	1.	$\{x + b > 0\}\ y := b\ \{x + y > 0\}$	AA
	2.	$\{a + b > 0\}\ x := a\ \{x + b > 0\}$	AA
	3.	$(a > 0) \wedge (b > 0) \to (a + b > 0)$	T
	4.	$\{(a > 0) \wedge (b > 0)\}\ x := a\ \{x + b > 0\}$	2, 3, Consequence
		QED	1, 4, Composition.

3. Use the composition rule (8.12) applied to a sequence of three statements.

a.	1.	$\{\text{temp} < x\}\ y := \text{temp}\ \{y < x\}$	AA
	2.	$\{\text{temp} < y\}\ x := y\ \{\text{temp} < x\}$	AA
	3.	$\{x < y\}\ \text{temp} := x\ \{\text{temp} < y\}$	AA
		QED	3, 2, 1, Composition.

4. a. First, prove the correctness of the wff $\{(x < 10) \wedge (x \geq 5)\}\ x := 4\ \{x < 5\}$:

1.	$\{4 < 5\}\ x := 4\ \{x < 5\}$	AA
2.	$(x < 10) \wedge (x \geq 5) \to (4 < 5)$	T
3.	$\{(x < 10) \wedge (x \geq 5)\}\ x := 4\ \{x < 5\}$	1, 2, Consequence
	QED.	

Second, prove that $(x < 10) \wedge \neg\,(x \geq 5) \to (x < 5)$. This is a valid wff because of the equivalence $\neg\,(x \geq 5) \equiv x < 5$. Thus the original wff is correct, by the if-then rule.

c. First, prove $\{\text{true} \wedge (x < y)\}\ x := y\ \{x \geq y\}$:

1.	$\{y \geq y\}\ x := y\ \{x \geq y\}$	AA
2.	$\text{true} \wedge (x < y) \to (y \geq y)$	T
3.	$\{\text{true} \wedge (x < y)\}\ x := y\ \{x \geq y\}$	1, 2, Consequence
	QED.	

Second, prove that $\text{true} \wedge \neg\,(x < y) \to (x \geq y)$. This is a valid wff because of the equivalence $\text{true} \wedge \neg\,(x < y) \equiv \neg\,(x < y) \equiv (x \geq y)$. Thus the original wff is correct, by the if-then rule.

5. a. Use the if-then-else rule. Thus we must prove the two statements

$$\{\text{true} \wedge (x < y)\}\ \text{max} := y\ \{(\text{max} \geq x) \wedge (\text{max} \geq y)\}$$
$$\{\text{true} \wedge (x \geq y)\}\ \text{max} := x\ \{(\text{max} \geq x) \wedge (\text{max} \geq y)\}.$$

For example, the first statement can be proved as follows:

1.	$\{(y \geq x) \wedge (y \geq y)\}$ max $:= y$ $\{(\max \geq x) \wedge (\max \geq y)\}$	AA
2.	true $\wedge\ (x < y)$	P
3.	$x < y$	2, Simp
4.	$x \leq y$	3, Add
5.	$y \geq y$	T
6.	$(y \geq x) \wedge (y \geq y)$	4, 5, Conj
7.	true $\wedge\ (x < y) \rightarrow (y \geq x) \wedge (y \geq y)$	2, 6, CP
8.	$\{$true $\wedge\ (x < y)\}$ max $:= y$ $\{(\max \geq x) \wedge (\max \geq y)\}$	1, 7, Consequence
	QED.	

6. a. The wff is incorrect if $x = 1$.

7. Since the wff fits the form of the while rule, we need to prove the following statement:

$$\{(x \geq y) \wedge \text{even}(x - y) \wedge (x \neq y)\} x := x - 1; y := y + 1 \{(x \geq y) \wedge \text{even}(x - y)\}.$$

Proof:

1.	$\{(x \geq y + 1) \wedge \text{even}\,(x - y - 1)\}$ $y := y + 1\, \{(x \geq y) \wedge \text{even}\ (x - y)\}$	AA
2.	$\{(x - 1 \geq y + 1) \wedge \text{even}\,(x - 1 - y - 1)\}$ $x := x - 1\, \{(x \geq y + 1) \wedge \text{even}\,(x - y - 1)\}$	AA
3.	$(x \geq y) \wedge \text{even}(x - y) \wedge (x \neq y)$	P
4.	$x \geq y + 2$	3, T
5.	$x - 1 \geq y + 1$	4, T
6.	$\text{even}(x - 1 - y - 1)$	3, T
7.	$(x - 1 \geq y + 1) \wedge \text{even}(x - 1 - y - 1)$	5, 6, Conj
8.	$(x \geq y) \wedge \text{even}(x - y) \wedge (x \neq y)$ $\rightarrow (x - 1 \geq y + 1) \wedge \text{even}(x - 1 - y - 1)$	3, 7, CP
	QED	1, 2, 8, Consequence, Composition.

Now the result follows from the while rule.

8. a. The postcondition $i = \text{floor}(x)$ is equivalent to $(i \leq x) \wedge (x < i + 1)$. This statement has the form $Q \wedge \neg\, C$, where C is the condition of the while loop and Q is the suggested loop invariant. To show that the while loop is correct with respect to Q, show that $\{Q \wedge C\}\ i := i + 1\ \{Q\}$ is correct. Once this is done, show that $\{x \geq 0\}\ i := 0\ \{Q\}$ is correct.

c. The given wff fits the form of the if-then-else rule. Therefore, we need to prove the following two wffs:

$$\{\text{true} \wedge (x \geq 0)\}\ S_1\ \{i = \text{floor}(x)\} \text{ and } \{\text{true} \wedge (x < 0)\}\ S_2\ \{i = \text{floor}(x)\}.$$

These two wffs are equivalent to the two wffs of parts (a) and (b). Therefore, the given wff is correct.

9. Let Q be the suggested loop invariant. Then the postcondition is equivalent to $Q \wedge \neg\ C$, where C is the while loop condition. Therefore, the program can be proven correct by proving the validity of the following two wffs:

$$\{Q \wedge C\}\ i := i + 1;\ s := s + i\ \{Q\} \text{ and } \{n \geq 0\}\ i := 0;\ s := 0;\ \{Q\}.$$

11. Letting Q denote the loop invariant, the while loop can be proved correct with respect to Q by proving the following wff:

$$\{Q \wedge (x \neq y)\}\ \textbf{if}\ x > y\ \textbf{then}\ x := x - y\ \textbf{else}\ y := y - x\ \{Q\}.$$

The parts of the program before and after the while loop can be proved correct by proving the following two wffs:

$$\{(a > 0) \wedge (b > 0)\}\ x := a;\ y := b\ \{Q\},$$
$$\{Q \wedge \neg\ (x \neq y)\}\ \text{great} := x\ \{\gcd(a, b) = \text{great}\}.$$

13. a. $\{(\text{if } j = i - 1 \text{ then } 24 \text{ else } a[j]) = 24\}$.

c. We obtain the precondition

$$\{((\text{if } i = j - 1 \text{ then } 12 \text{ else } (\text{if } i = i + 1 \text{ then } 25 \text{ else } a[i])) = 12)$$
$$\wedge\ ((\text{if } j = j - 1 \text{ then } 12 \text{ else } (\text{if } j = i + 1 \text{ then } 25 \text{ else } a[j])) = 25)\}.$$

Since it is impossible to have $i = i + 1$ and $j = j - 1$, the precondition can be simplified to

$$\{((\text{if } i = j - 1 \text{ then } 12 \text{ else } a[i]) = 12) \wedge ((\text{if } j = i + 1 \text{ then } 25 \text{ else } a[j]) = 25)\}.$$

14. a.

1.	$\{(\text{if } j = i - 1 \text{ then } 24 \text{ else a}[j]) = 24\}$ $a[i - 1] := 24\ \{a[j] = 24\}$	AAA
2.	$(i = j + 1) \wedge (a[j] = 39)$	P
3.	$i = j + 1$	2, Simp
4.	$24 = 24$	T
5.	$(i = j + 1) \rightarrow (24 = 24)$	T (true conclusion)
6.	$(i \neq j + 1) \rightarrow (a[j] = 24)$	3, T (false premise)
7.	$(\text{if } j = i - 1 \text{ then } 24 \text{ else a}[j]) = 24$	5, 6, Conj, T
8.	$(i = j + 1) \wedge (a[j] = 39) \rightarrow$ $((\text{if } j = i - 1 \text{ then } 24 \text{ else a}[j]) = 24)$	2, 7, CP
	QED	1, 8, Consequence.

c. 1. $\{((\text{if } i = j - 1 \text{ then } 12 \text{ else } a[i]) = 12) \wedge$
$((\text{if } j = j - 1 \text{ then } 12 \text{ else } a[j]) = 25)\}$
$a[j - 1] := 12$
$\{(a[i] = 12) \wedge (a[j] = 25)\}$ AAA

2. $\{((\text{if } i = j - 1 \text{ then } 12 \text{ else } a[i]) = 12) \wedge (a[j] = 25)\}$
$a[j - 1] := 12$
$\{(a[i] = 12) \wedge (a[j] = 25)\}$ 1, T

3. $\{((\text{if } i = j - 1 \text{ then } 12 \text{ else}$
$(\text{if } i = i + 1 \text{ then } 25 \text{ else } a[i])) = 12)$
$\wedge ((\text{if } j = i + 1 \text{ then } 25 \text{ else } a[j]) = 25)\}$
$a[i + 1] := 25$
$\{((\text{if } i = j - 1 \text{ then } 12 \text{ else } a[i]) = 12) \wedge (a[j]) = 25)\}$ AAA

4. $\{((\text{if } i = j - 1 \text{ then } 12 \text{ else } a[i]) = 12)$
$\wedge ((\text{if } j = i + 1 \text{ then } 25 \text{ else } a[j]) = 25)\}$
$a[i + 1] := 25$
$\{((\text{if } i = j - 1 \text{ then } 12 \text{ else } a[i]) = 12) \wedge (a[j]) = 25)\}$ 3, T

5. $(i = j - 1) \wedge (a[i] = 25) \wedge (a[j] = 12)$ P

6. $i = j - 1$ 2, Simp

7. $((\text{if } i = j - 1 \text{ then } 12 \text{ else } a[i]) = 12)$
$\wedge ((\text{if } j = i + 1 \text{ then } 25 \text{ else } a[j]) = 25)$ 6, T

8. $(i = j - 1) \wedge (a[i] = 25) \wedge (a[j] =12)$
$\rightarrow ((\text{if } i = j - 1 \text{ then } 12 \text{ else } a[i]) = 12)$
$\wedge ((\text{if } j = i + 1 \text{ then } 25 \text{ else } a[j]) = 25)$ 5, 7, CP

QED 2, 4, Consequence, Composition.

15. a. After applying AAA to the postcondition and assignment, we obtain the condition even$(a[i] + 1)$. It is clear that the precondition even$(a[i])$ does not imply even$(a[i] + 1)$. **c.** After applying AAA twice to the postcondition and two assignments, we obtain the condition

$$\forall j\ ((1 \leq j \leq 5) \rightarrow (\text{if } j = 3 \text{ then } 355 \text{ else } a[j]) = 23).$$

This wff is the conjunction of five propositions, one for each j, where $1 \leq j \leq 5$. For $j = 3$ we obtain the proposition

$$((1 \leq 3 \leq 5) \rightarrow (\text{if } 3 = 3 \text{ then } 355 \text{ else } a[3]) = 23),$$

which is equivalent to the false statement $(1 \leq 3 \leq 5) \rightarrow (355 = 23)$. Therefore, the given precondition cannot imply the obtained condition.

16. a. Define $f(i, x) = x - i$. If $s = (i, x)$, then after the execution of the loop body the state will be $t = (i + 1, x)$. Thus $f(s) = x - i$ and $f(t) = x - i - 1$. To prove termination, assume P and C are true and prove that $f(s)$, $f(t) \in \mathbb{N}$ and $f\ (s) > f(t)$. So assume $\text{int}(i) \wedge (\text{int}(x)) \wedge i \leq x$ and $i < x$. It follows that i and x are integers and $i < x$. So $x - i$ is a positive integer and $x - i - 1$ is a nonnegative integer. In other words, both $x - i$ and $x - i - 1$ are natural numbers, which tells us that $f(s), f(t) \in \mathbb{N}$. Since subtraction by 1 yields a smaller number we have $x - i > x - i - 1$, so that $f(s) > f(t)$. Therefore, the loop terminates.
c. Define $f(x) = \mid x \mid$. If $s = x$, then after the execution of the loop body the state will be $t = x/2$. So $f(s) = \mid x \mid$ and $f(t) = \mid x/2 \mid$. To prove termination, assume P and C are true and prove that $f(s)$, $f(t) \in \mathbb{N}$ and $f(s) > f(t)$. So assume $\text{int}(x)$ and $\text{even}(x) \wedge x \neq 0$. It follows that x is a nonzero even integer. Since x is even, it is divisible by 2. So $x/2$ is still an integer. Thus $\mid x \mid$ and $\mid x/2 \mid$ are both natural numbers, so we have $f(s)$, fx is nonzero it follows that $\mid x \mid > \mid x/2 \mid$, so that $f(s) > f(t)$. Therefore, the loop terminates.

17. a. We are given that $f(x, y) = x + y$ and $W = \mathbb{N}$. To prove termination, assume P and C are true and prove that $f(s)$, $f(t) \in \mathbb{N}$ and $f(s) > f(t)$. So assume $\text{pos}(x) \wedge \text{pos}(y)$ and $x \neq y$. If $s = (x, y)$, then the state after the execution of the loop body will depend on whether $x < y$. If $x < y$, then $t = (x, y - x)$, which gives $f(t) = x$. Otherwise, if $x > y$, then $t = (x - y, y)$, which gives $f(t) = y$. Since x and y are positive integers, it follows that both $x + y$ and x are natural numbers and $x + y > x$ and $x + y > y$. So in either case (i.e., $x < y$ or $x > y$) we have $f(s)$, $f(t) \in \mathbb{N}$ and $f(s) > f(t)$. Therefore, the loop terminates.
c. We are given that $f(x, y) = (x, y)$ and $W = \mathbb{N} \times \mathbb{N}$ with the lexicographic ordering. To prove termination, assume P and C are true and prove that $f(s)$, $f(t) \in \mathbb{N}$ and $f(s) > f(t)$. So assume $\text{pos}(x) \wedge \text{pos}(y)$ and $x \neq y$. If $s = (x, y)$, then the state t after the execution of the loop body has two possible values. If $x < y$, then $t = (x, y - x)$, so it follows that $f(s)$, $f(t) \in W$ and we also have

$$f(s) = f(x, y) = (x, y) \succ (x, y - x) = f(x, y - x) = f(t).$$

If $x > y$, then $t = (x - y, y)$, so it follows that $f(s)$, $f(t) \in W$ and we also have

$$f(s) = f(x, y) = (x, y) \succ (x - y, y) = f(x - y, y) = f(t).$$

Therefore, the loop terminates.

18. a. The definition $f(x, y) = \mid x - y \mid$ cannot be used because there are state values s and t such that $f(s) \leq f(t)$, which is contrary to the need in (8.19) for $f(s) > f(t)$. For example, if $s = (x, y) = (10, 13)$, then $f(s) = 3$. But after the body of the loop executes, we have $t = (x, y - x) = (10, 3)$, which gives $f(t) = 7$.

19. Let P be the loop invariant, $P = \exists x\ (a = xb + r) \wedge (0 \leq r)$. The postcondition $r = a \bmod b$ means that r is the remainder obtained on divsion of a by b, where $0 \leq r \leq b$. This is exactly the condtion $P \wedge \neg\ (r \geq b)$ which is needed for the end of the while loop. So the proof of partial correctness follows by composition from the correctness of the following two statements.

1. $\{(a \geq 0) \wedge (b > 0)\}\ r := a\ \{P\}$.

2. $\{P\}$ **while** $r \geq b$ **do** $r := r - b$ **od** $\{P \wedge \neg\ (r \geq b)\}$.

Proof of Statement 1.

1.	$\{\exists x\ (a = xb + a) \wedge (0 \leq a)\}\ r := a\ \{P\}$	AA
2.	$(a \geq 0) \wedge (b > 0)$	P
3.	$a = (0)b + a$	T
4.	$\exists x\ (a = xb + a)$	3, EG
5.	$a \geq 0$	2, Simp
6.	$\exists\ x\ (a = xb + a) \wedge (0 \leq a)$	4, 5, Conj
7.	$(a \geq 0) \wedge (b > 0) \rightarrow \exists x\ (a = xb + a) \wedge (0 \leq a)$	2, 6, CP
8.	$\{(a \geq 0) \wedge (b > 0)\}\ r := a\ \{P\}$	1, 7, Consequence
	QED.	

Proof of Statement 2.

1.	$\{\exists x\ (a = xb + r - b) \wedge (0 \leq r - b)\}\ r := r - b\ \{P\}$	AA
2.	$\exists x\ (a = xb + r) \wedge (0 \leq a) \wedge (r \geq b)$	P
3.	$\exists x\ (a = xb + r)$	2, Simp
4.	$a = qb + r$	3, EI
5.	$a = (q + 1)b + r - b$	4, T
6.	$\exists x\ (a = xb + r - b)$	5, EG
7.	$r \geq b$	2, Simp
8.	$0 \leq r - b$	7, T
9.	$\exists x\ (a = xb + r - b) \wedge (0 \leq r - b)$	6, 8, Conj
10.	$\exists x\ (a = xb + r) \wedge (0 \leq a) \wedge (r \geq b)$ $\rightarrow \exists x\ (a = xb + r - b) \wedge (0 \leq r - b)$	2, 9, CP
11.	$\{P \wedge r \geq b\}\ r := r - b\ \{P\}$	1, 10, Consequence
12.	$\{P\}$ **while** $r \geq b$ **do** $r := r - b$ **od** $\{P \wedge \neg\ (r \geq b)\}$	11, While-rule
	QED.	

Proof of Termination.

Let $W = \mathbb{N}$ with the usual ordering and let $f(a, b, r) = r$. If $s = (a, b, r)$, then the state t after the execution of the loop body is $t = (a, b, r - b)$. To prove termination, assume P and C are true and prove that $f(s), f(t) \in \mathbb{N}$ and $f(s) > f(t)$. So assume $\exists x\ (a = xb + r) \wedge (0 \leq r)$ and $(r \geq b)$. It follows that $r \geq 0$ and also $r - b \geq 0$, so we have $f(s), f(t) \in \mathbb{N}$. Since $b > 0$, it follows that $f(s) > f(t)$. Therefore, the loop terminates.

Section 8.3

1. a. Second. **c.** Fifth. **e.** Third. **g.** Third. **i.** Fourth.

2. a. $\exists A \, \exists \, B \, \forall x \, \neg \, (A(x) \wedge B(x))$.

3. a. Let S be state and C be city. Then $\forall S \, \exists C \, (S(C) \wedge (C = \text{Springfield}))$. The wff is second order.

c. Let H, R, S, B, and A mean house, room, shelf, book, and author. Then $\exists \, H \, \exists R \, \exists S \, \exists B \, (H(R) \wedge R(S) \wedge S(B) \wedge A(B, \text{Thoreau}))$. The wff is fourth order.

e. The statement can be expressed as follows:
$\exists S \, \exists A \, \exists B \, (\forall x \, (A(x) \vee B(x) \to S(x)) \wedge \forall x \, (S(x) \to A(x) \vee B(x)) \wedge \forall x \, \neg \, (A(x) \wedge B(x)))$. The wff is second order.

5. $\forall R \, (B(R) \to (\forall x \, \neg \, R(x, x) \wedge \forall \, x \, \forall y \, \forall z \, (R(x, y) \wedge R(y, z) \to R(x, z)) \to \forall x \, \forall y \, (R(x, y) \to \neg \, R(y, x))))$.

7. Think of $S(x)$ as $x \in S$. **a.** For any domain D the antecedent is false because S can be the empty set. Thus the wff is true for all domains.

c. For any domain D the consequent is true because S can be chosen as D. Thus the wff is true for all domains.

8. Informal proof: Let I be an interpretation with domain D. Then wff has the following meaning with respect to I. For every subset P of D there is a subset Q of D such that $x \in Q$ implies $x \in P$. This statement is true because we can choose Q to be P. So I is a model for the wff. Since I was an arbitrary interpretation, it follows that the wff is valid. QED.

In the formal proof, we'll represent instantiations of the variables P and Q with lower-case p and q.

Formal proof:

1.	$\neg \, \forall P \, \exists \, Q \, \forall x \, (Q(x) \to P(x))$	P for IP
2.	$\exists P \, \forall Q \, \exists x \, (Q(x) \wedge \neg \, P(x))$	1, T
3.	$\forall Q \, \exists x \, (Q(x) \wedge \neg \, p(x))$	2, EI
4.	$\exists x \, (p(x) \wedge \neg \, p(x))$	3, UI
5.	$p(c) \wedge \neg \, p(c)$	4, EI
6.	false	5, T
	QED	1, 6, IP.

9. a. Assume that the statement is false. Then there is some line L containing every point. Now Axiom 4 says that there are three distinct points not on the same line. This is a contradiction. Thus the statement is true.

c. Let w be a point. By Axiom 4 there is another point x such that $x \neq w$. By Axiom 1 there is a line L on x and w. By part (a) there is a point z not on L. By Axiom 1 there is a line M on w and z. Since z is on M and z is not on L, it follows that $L \neq M$.

10. Here are some sample formalizations.

a. $\forall L \, \exists x \, \neg \, L(x)$.

Proof:

1.	$\neg \forall L \exists x \neg L(x)$	P for IP
2.	$\exists L \forall x L(x)$	1, T
3.	$\forall x\ l(x)$	2, EI
4.	Axiom 3	
5.	$l(a) \wedge l(b) \to \neg l(c)$	4, EI, EI, EI, Simp, UI
6.	$l(a) \wedge l(b)$	3, UI, UI, Conj
7.	$\neg l(c)$	5, 6, MP
8.	$l(c)$	3, UI
9.	false	7, 8, Conj, T
	QED	1, 9, IP.

c. $\forall x \exists L \exists M (L(x) \wedge M(x) \wedge \exists y (\neg L(y) \wedge M(y)))$.

Proof:

1.	$\neg (\forall x \exists L \exists M (L(x) \wedge M(x) \wedge \exists y (\neg L(y) \wedge M(y))))$	P for IP
2.	$\exists x \forall L \forall M (\neg (L(x) \wedge M(x)) \vee \forall y (L(y) \vee \neg M(y)))$	1, T
3.	$\forall L \forall M (\neg (L(a) \wedge M(a)) \vee \forall y (L(y) \vee \neg M(y)))$	2, EI, UI, UI
4.	$(b \neq c) \wedge (b \neq d) \wedge (c \neq d)$ $\wedge \forall L (L(b) \wedge L(c) \to \neg L(d))$	Axiom 4, EI, EI, EI
5.	$a = b$	P for IP
6.	$b \neq c$	4, Simp
7.	$a \neq c$	5, 6, EE
8.	$(a \neq c) \to \exists L (L(a) \wedge L(c))$	Axiom 1, UI, UI
9.	$\exists L (L(a) \wedge L(c))$	7, 8, MP
10.	$l(a) \wedge l(c)$	9, EI
11.	$b \neq d$	4, Simp
12.	$a \neq d$	5, 11, EE
13.	$(a \neq d) \to \exists L (L(a) \wedge L(d))$	Axiom 1, UI, UI
14.	$\exists L (L(a) \wedge L(d))$	12, 13, MP
15.	$m(a) \wedge m(d)$	14, EI
16.	$\neg (l(a) \wedge m(a)) \vee \forall y (l(y) \vee \neg m(y))$	3, UI, UI
17.	$l(a) \wedge m(a)$	10, 15, Simp, Conj
18.	$\forall y (l(y) \vee \neg m(y))$	16, 17, DS
19.	$l(d) \vee \neg m(d)$	18, UI
20.	$l(d)$	15, Simp, 19, DS
21.	$\forall L (L(b) \wedge L(c) \to \neg L(d))$	4, Simp
22.	$l(b) \wedge l(c) \to \neg l(d)$	21, UI
23.	$l(b) \wedge l(c)$	5, 10, EE
24.	$\neg l(d)$	22, 23, MP
25.	false	20, 24, Conj, T

26.	$a \neq b$	5, 25, IP
27.	$a \neq c$	T (like $a \neq b$)
28.	$a \neq d$	T (like $a \neq b$)
29.	$(a \neq b) \rightarrow \exists L\,(L\,(a) \wedge L\,(b))$	Axiom 1, UI, UI
30.	$\exists L\,(L\,(a) \wedge L\,(b))$	26, 29, MP
31.	$l\,(a) \wedge l\,(b)$	30, EI
32.	$(a \neq c) \rightarrow \exists L\,(L\,(a) \wedge L\,(c))$	Axiom 1, UI, UI
33.	$\exists L\,(L\,(a) \wedge L\,(c))$	27, 32, MP
34.	$m\,(a) \wedge m\,(c)$	33, EI
35.	$(a \neq d) \rightarrow \exists L\,(L\,(a) \wedge L\,(d))$	Axiom 1, UI, UI
36.	$\exists L\,(L\,(a) \wedge L\,(d))$	28, 35, MP
37.	$n\,(a) \wedge n\,(d)$	36, EI
38.	$l\,(a) \wedge m\,(a)$	31, Simp, 34, Simp, Conj
39.	$\neg\,(l\,(a) \wedge m\,(a)) \vee \forall y\,(l\,(y) \vee \neg\, m\,(y))$	3, UI, UI
40.	$\forall y\,(l\,(y) \vee \neg\, m\,(y))$	38, 39, DS
41.	$l\,(c) \vee \neg\, m\,(c)$	40, UI
42.	$l\,(c)$	34, Simp, 41, DS
43.	$l\,(a) \wedge n\,(a)$	31, Simp, 37, Simp, Conj
44.	$\neg\,(l\,(a) \wedge n\,(a)) \vee \forall y\,(l\,(y) \vee \neg\, n\,(y))$	3, UI, UI
45.	$\forall y\,(l\,(y) \vee \neg\, n\,(y))$	43, 44, DS
46.	$l\,(d) \vee \neg\, n\,(d)$	45, UI
47.	$l\,(d)$	37, Simp, 46, DS
48.	$l\,(b) \wedge l\,(c)$	31, Simp, 42, Conj
49.	$\forall L\,(L\,(b) \wedge L\,(c) \rightarrow \neg\, L\,(d))$	4, Simp
50.	$l\,(b) \wedge l\,(c) \rightarrow \neg\, l\,(d)$	49, UI
51.	$\neg\, l\,(d)$	48, 50, MP
52.	false	47, 51, Conj, T
	QED	1, 32, IP.

Chapter 9

Section 9.1

1. a. $(A \lor C \lor D) \land (B \lor C \lor D)$.
c. $\forall x\ (\neg\, p(x, c) \lor q(x))$.
e. $\forall x\ \forall y\ (p(x, y) \lor q(x, y, f(x, y)))$.

2. $p \lor \neg\, p$ and $p \lor \neg\, p \lor q \lor q$.

3. a.

1.	$A \lor B$	P
2.	$\neg\, A$	P
3.	$\neg\, B \lor C$	P
4.	$\neg\, C$	P
5.	B	1, 2, R
6.	$\neg\, B$	3, 4, R
7.	$\square$	5, 6, R.

c.

1.	$A \lor B$	P
2.	$A \lor \neg\, C$	P
3.	$\neg\, A \lor C$	P
4.	$\neg\, A \lor \neg\, B$	P
5.	$C \lor \neg\, B$	P
6.	$\neg\, C \lor B$	P
7.	$B \lor C$	1, 3, R
8.	$B \lor B$	6, 7, R
9.	$\neg\, A$	4, 8, R
10.	$\neg\, C$	2, 9, R
11.	$\neg\, B$	5, 10, R
12.	A	1, 11, R
13.	$\square$	9, 12, R.

4. a. $\{y/x\}$. **c.** $\{y/a\}$. **e.** $\{x/f(a),\ y/f(b),\ z/b\}$.

5. a. $\{x/f(a, b),\ v/f(y, a),\ z/y\}$ or $\{x/f(a, b),\ v/f(z, a),\ y/z\}$.
c. $\{x/g(a),\ z/g(b),\ y/b\}$.

6. a. $\{x/f(a, b),\ v/f(z, a),\ y/z\}$. **c.** $\{x/g(a),\ z/g(b),\ y/b\}$.

7. Make sure the clauses to be resolved have distinct sets of variables. The answers are $p(x) \lor \neg\, p(f(a))$ and $p(x) \lor \neg\, p(f(a)) \lor q(x) \lor q(f(a))$.

8. a.

1.	$p(x)$	P
2.	$q(y, a) \lor \neg\, p(a)$	P
3.	$\neg\, q(a, a)$	P
4.	$\neg\, p(a)$	2, 3, R, $\{y/a\}$
5.	$\square$	1, 4, R, $\{x/a\}$.
	QED	

c.	1.	$p(a) \vee p(x)$	P
	2.	$\neg\, p(a) \vee \neg\, p(y)$	P
	3.	$\square$	1, 2, R, $\{x/a,\ y/a\}$.
		QED	

e. Number the clauses 1, 2, and 3. Resolve 2 with 3 by unifying all four of the p atoms to obtain the clause $\neg\, q(a) \vee \neg\, q(a)$. Resolve this clause with 1 to obtain the empty clause.

9. a. After negating the statement and putting the result in clausal form, we obtain the following proof:

1.	$A \vee B$	P
2.	$\neg\, A$	P
3.	$\neg\, B$	P
4.	B	1, 2, R
5.	$\square$	3, 4, R. QED

c. After negating the statement and putting the result in clausal form, we obtain the following proof:

1.	$p \vee q$	P
2.	$\neg\, q \vee r$	P
3.	$\neg\, r \vee s$	P
4.	$\neg\, p$	P
5.	$\neg\, s$	P
6.	$\neg\, r$	3, 5, R
7.	$\neg\, q$	2, 6, R
8.	q	1, 4, R
9.	$\square$	7, 8, R. QED

10. a. After negating the statement and putting the result in clausal form, we obtain the following proof:

1.	$p(x)$	P
2.	$\neg\, p(y)$	P
3.	$\square$	1, 2, R, $\{x/y\}$. QED

c. After negating the statement and putting the result in clausal form, we obtain the following proof:

1.	$p(x, a)$	P
2.	$\neg\, p(b, y)$	P
3.	$\square$	1, 2, R, $\{x/b,\ y/a\}$. QED

e. After negating the statement and putting the result in clausal form, we obtain the following proof:

1.	$p(x) \vee q(y)$	P
2.	$\neg\, p(a)$	P
3.	$\neg\, q(a)$	P
4.	$q(y)$	1, 2, R, $\{x/a\}$
5.	$\square$	3, 4, R, $\{y/a\}$. QED

11. a. We need to show that $x(\theta\sigma) = (x\theta)\sigma$ for each variable x in E. First, suppose $x/t \in \theta$ for some term t. If $x = t\sigma$, then $x(\theta\sigma) = x$ because the binding $x/t\sigma$ has been removed from $\theta\sigma$. But since $x/t \in \theta$, it follows that $x\theta = t$. Now apply σ to both sides to obtain $(x\theta)\sigma = t\sigma = x$. Therefore, $x(\theta\sigma) = x = (x\theta)\sigma$. If $x \neq t\sigma$, then $x(\theta\sigma) = t\sigma = (x\theta)\sigma$. Second, suppose that $x/t \in \sigma$ and x does not occur as a numerator of θ . Then $x(\theta\sigma) = t = x\sigma = (x\theta)\sigma$. Lastly, if x does not occur as a numerator of either σ or θ, then the substitutions have no effect on x. Thus $x(\theta\sigma) = x = (x\theta)\sigma$.
c. If $x/t \in \theta$, then $x/t = x/t\epsilon$, so it follows from the definition of composition that $\theta = \theta\epsilon$. For any variable x we have $x(\epsilon\theta) = (x\epsilon)\theta = x\theta$. Therefore, $\theta\epsilon = \epsilon\theta = \theta$.
e. $(A \cup B)\theta = \{E\theta \mid E \in A \cup B\} = \{E\theta \mid E \in A\} \cup \{E\theta \mid E \in B\} = A\theta \cup B\theta$.
12. a. In first-order predicate calculus the argument can be written as

$$\forall x\ (C(x) \to P(x)) \wedge \exists x\ (C(x) \wedge L(x)) \to \exists\ x\ (P(x) \wedge L(x)),$$

where $C(x)$ means that x is a computer science major, $P(x)$ means that x is a person, and $L(x)$ means that x is a logical thinker. After negating the wff and transforming the result into clausal form, we obtain the proof:

1.	$\neg\, C(x) \vee P(x)$	P
2.	$C(a)$	P
3.	$L(a)$	P
4.	$\neg\, P(z) \vee \neg\, L(z)$	P
5.	$\neg\, P(a)$	3, 4, R, $\{z/a\}$
6.	$\neg\, C(a)$	1, 5, R, $\{x/a\}$
7.	$\square$	2, 6, R, $\{\ \}$. QED

13. a. Let $D(x)$ mean that x is a dog, $L(x)$ mean that x likes people, $H(x)$ mean that x hates cats, and a = Rover. Then the argument can be formalized as follows:

$$\forall x\ (D(x) \to L(x) \vee H(x)) \wedge D(a) \wedge \neg\, H(a) \to \exists x\ (D(x) \wedge L(x)).$$

After negating the wff and transforming the result into clausal form, we obtain the proof:

1.	$\neg D(x) \lor L(x) \lor H(x)$	P
2.	$D(a)$	P
3.	$\neg H(a)$	P
4.	$\neg D(y) \lor \neg L(y)$	P
5.	$L(a) \lor H(a)$	1, 2, R, $\{x/a\}$
6.	$L(a)$	3, 5, R, $\{\ \}$
7.	$\neg D(a)$	4, 6, R, $\{y/a\}$
8.	$\square$	2, 7, R, $\{\ \}$. QED

c. Let $H(x)$ mean that x is a human being, $Q(x)$ mean that x is a quadruped, and $M(x)$ mean that x is a man. Then the argument can be formalized as

$$\forall x\ (H(x) \to \neg Q(x)) \land \forall x\ (M(x) \to H(x)) \to \forall x\ (M(x) \to \neg Q(x)).$$

After negating the wff and transforming the result into clausal form, we obtain the proof:

1.	$\neg H(x) \lor \neg Q(x)$	P
2.	$\neg M(y) \lor H(y)$	P
3.	$M(a)$	P
4.	$Q(a)$	P
5.	$H(a)$	2, 3, R, $\{y/a\}$
6.	$\neg Q(a)$	1, 5, R, $\{x/a\}$
7.	$\square$	4, 6, R, $\{\ \}$. QED

e. Let $F(x)$ mean that x is a freshman, $S(x)$ mean that x is a sophomore, $J(x)$ mean that x is a junior, and $L(x, y)$ mean that x likes y. Then the argument can be formalized as $A \to B$, where

$$A = \exists x\ (F(x) \land \forall y\ (S(y) \to L(x, y))) \land \forall x\ (F(x) \to \forall y\ (J(y) \to \neg L(x, y)))$$

and $B = \forall x\ (S(x) \to \neg J(x))$. After negating the wff and transforming the result into clausal form, we obtain the proof:

1.	$F(a)$	P
2.	$\neg S(x) \lor L(a, x)$	P
3.	$\neg F(y) \lor \neg J(z) \lor \neg L(y, z)$	P
4.	$S(b)$	P
5.	$J(b)$	P
6.	$\neg J(z) \lor \neg L(a, z)$	1, 3, R, $\{y/a\}$
7.	$\neg L(a, b)$	5, 6, R, $\{z/b\}$
8.	$\neg S(b)$	2, 7, R, $\{x/b\}$
9.	$\square$	4, 8, R, $\{\ \}$. QED

14. a. Here is an indirect proof that W is valid.

Proof:	1.	$\neg \forall x\ \exists y\ (p(x, y) \vee \neg p(y, y))$	P for IP
	2.	$\exists x\ \forall y\ (\neg p(x, y) \wedge p(y, y))$	1, T
	3.	$\forall y\ (\neg p(c, y) \wedge p(c, c))$	2, EI
	4.	$\neg p(c, c) \wedge p(c, c)$	3, UI
	5.	false	4, T
		QED	1, 5, IP.

15. a. Let I be an interpretation for W. If C is true for I, then $\exists y\ (p(y) \wedge \neg C)$ is false, so W is false. If C is false for I, then W becomes $(\exists x\ p(x) \rightarrow \text{false}) \wedge \exists\ y\ (p(y) \wedge \neg\ \text{false}) \equiv \neg\ \exists x\ p(x) \wedge \exists y\ p(y)$, which is false. Therefore, W is false for I. Since I was arbitrary, W is unsatisfiable.

c. After eliminating $\rightarrow$ from W, we apply Skolem's rule to obtain the wff $(\neg p(a) \vee C) \wedge (p(b) \wedge \neg C)$. Define an interpretation for this wff by letting $C =$ false, $p(a) =$ false, and $p(b) =$ true. This interpretation makes the wff true. So it is satisfiable.

Section 9.2

1. a. isChildOf(x, y) $\leftarrow$ isParentOf(y, x).

c. isGreatGrandParentOf(x, y)
$\leftarrow$ isParentOf(x, w), isParentOf(w, z), isParentOf(z, y).

2. a. The following definition will work if $x \neq y$:

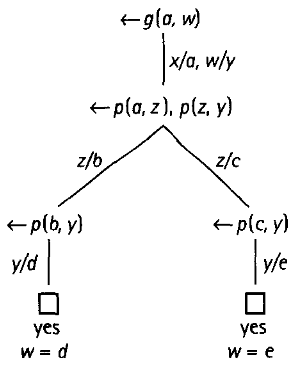
isSiblingOf(x, y) $\leftarrow$ isParentOf(z, x), isParentOf(z, y).

c. Let s denote isSecondCousinOf. One possible definition is

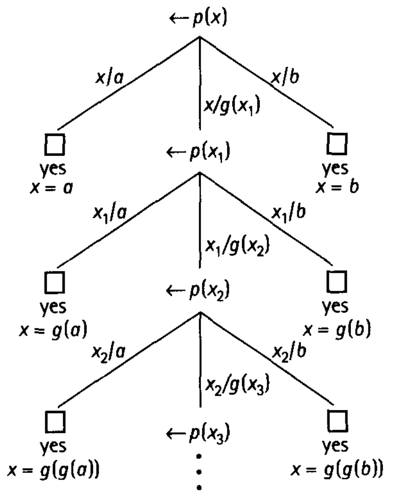
$s(x, y) \leftarrow$ isParentOf(z, x), isParentOf(w, y), isCousinOf(z, w).

3. a.	1.	$p(a, b)$	P
	2.	$p(a, c)$	P
	3.	$p(b, d)$	P
	4.	$p(c, e)$	P
	5.	$g(x, y) \leftarrow p(x, z), p(z, y)$	P
	6.	$\leftarrow g(a, w)$	P initial goal
	7.	$\leftarrow p(a, z), p(z, y)$	5, 6, R, $\theta_1 = \{x/a,\ w/y\}$.
	8.	$\leftarrow p(b, y)$	1, 7, R, $\theta_2 = \{z/b\}$
	9.	$\Box$	3, 8, R, $\theta_3 = \{y/d\}$. QED

b.



4. a.



c. $\{g^n(a) \mid n \in \mathbb{N}\}$.

5. a. The program returns the answer yes.

6. a. The symmetric closure s can be defined by the two clause program:

$$s(x, y) \leftarrow r(x, y).$$
$$s(x, y) \leftarrow r(y, x).$$

7. a. fib(0, 0).
fib(1, 1).
fib(x, $y + z$) $\leftarrow$ fib($x - 1$, y), fib($x - 2$, z).

c. pnodes($\langle\ \rangle$, 0).
pnodes($\langle L, a, R\rangle$, $1 + x + y$) $\leftarrow$ pnodes(L, x), pnodes(R, y).

8. a. equalLists($\langle\ \rangle$, $\langle\ \rangle$).
equalLists($x :: t$, $x :: s$) $\leftarrow$ equalLists(t, s).

c. all(x, $\langle\ \rangle$, $\langle\ \rangle$).
all(x, $x :: t$, u) $\leftarrow$ all(x, t, u)
all(x, $y :: t$, $y :: u$) $\leftarrow$ all(x, t, u).

e. subset($\langle\ \rangle$, y).
subset($x :: t$, y) $\leftarrow$ member(x, y), subset(t, y).

g. Using the "remove" predicate from Example 9.26, which removes one occurrence of an element from a list, the program to test for a subbag can be written as follows:

subBag($\langle\ \rangle$, y).
subBag($x :: t$, y) $\leftarrow$ member(x, y), remove(x, y, w), subBag(t, w).

9. Let the predicate schedule(L, S) mean that S is a schedule for the list of classes L. For example, if L = $\langle$english102, math200$\rangle$, then S is a list of 4-tuples of the form (name, section, time, place). For the example, S might look like the following:

$\langle$(english102, 2, 3pm, ivy238), (math200, 1, 10am, briar315)$\rangle$.

Assume that the available classes are listed as facts of the following form:

class(name, section, time, place).

The following solution will yield one schedule of classes that might contain time conflicts. All schedules can be found by backtracking. If a class cannot be found, a note is made to that effect.

```
schedule(⟨ ⟩,⟨ ⟩).
schedule(x :: y, S) ← class(x, Sect, Time, Place),
                      schedule(y, T),
                      cat(⟨(x, Sect, Time, Place)⟩, T, S).
schedule(x :: y, ⟨unfillable⟩).
```

10. Let letters(A, L) mean that L is the list of propositional letters that occur in the wff A. Let replace(p, true, A, B) mean $B = A(p/\text{true})$. Then we can start the process for a wff A with the goal $\leftarrow$ tautology(A, Answer), where A is a tautology if Answer = true. The initial definitions might go like the following, where uppercase letters denote variables:

```
tautology(A, Answer) ← letters(A, L), evaluate(A, L, Answer).
evaluate(A, ⟨ ⟩, Answer) ← value(A, Answer).
evaluate(A, H :: T, Answer) ← replace(H, true, A, B),
                              replace(H, false, A, C),
                              evaluate(B ∧ C, Answer).
```

When "value" is called, A is a proposition containing only true and false terms. The definition for the "replace" predicate might include some clauses like the following:

$$\begin{aligned}
&\text{replace}(X, \text{true}, X, \text{true}).\\
&\text{replace}(X, \text{true}, \neg X, \text{false}).\\
&\text{replace}(X, \text{true}, \neg A, \neg B) \leftarrow \text{replace}(X, \text{true}, A, B).\\
&\text{replace}(X, \text{true}, A \wedge X, B) \leftarrow \text{replace}(X, \text{true}, A, B).\\
&\text{replace}(X, \text{true}, X \wedge A, B) \leftarrow \text{replace}(X, \text{true}, A, B).\\
&\text{replace}(X, \text{true}, A \wedge B, C \wedge D) \leftarrow \text{replace}(X, \text{true}, A, C),\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{replace}(X, \text{true}, B, D).
\end{aligned}$$

Continue by writing the clauses for the false case and for the other operators $\vee$ and $\rightarrow$. The first few clauses for the "value" predicate might include some clauses like the following:

$$\begin{aligned}
&\text{value}(\text{true}, \text{true}).\\
&\text{value}(\text{false}, \text{false}).\\
&\text{value}(\neg\, \text{true}, \text{false}).\\
&\text{value}(\neg\, \text{false}, \text{true}).\\
&\text{value}(\neg X, Y) \leftarrow \text{value}(X, A), \text{value}\ (\neg A, Y).\\
&\text{value}(\text{false} \wedge X, \text{false}).\\
&\text{value}(X \wedge \text{false}, \text{false}).\\
&\text{value}(\text{true} \wedge X, Y) \leftarrow \text{value}(X, Y).\\
&\text{value}(X \wedge \text{true}, Y) \leftarrow \text{value}(X, Y).\\
&\text{value}(X \wedge Y, Z) \leftarrow \text{value}(X, U), \text{value}(Y, V), \text{value}(U \wedge V, Z).
\end{aligned}$$

Continue by writing the clauses to find the value of expressions containing the operators $\vee$ and $\rightarrow$. The predicate to construct the list of propositional letters in a wff might start off something like the following:

$$\begin{aligned}
&\text{letters}(X, \langle X\rangle) \leftarrow \text{atom}(X).\\
&\text{letters}(X \wedge Y, Z) \leftarrow \text{letters}(X, U), \text{letters}(Y, V), \text{cat}(U, V, Z).
\end{aligned}$$

Continue by writing the clauses for the other operations.

Chapter 10

Section 10.1

1. The zero is m because $\min(x, m) = \min(m, x) = m$ for all $x \in A$. The identity is n because $\min(x, n) = \min(n, x) = x$ for all $x \in A$. If $x, y \in A$ and $\min(x, y) = n$, then x and y are inverses of each other. Since n is the largest element of A, it follows that n is the only element with an inverse.

2. a. No; no; no. **c.** True; false; false is its own inverse.

3. $S = \{a, f(a), f^2(a), f^3(a), f^4(a)\}$.

4. a. An element z is a zero if both row z and column z contain only the element z. **c.** If x is an identity, then an element y has a right and left inverse w if x occurs in row y column w and also in row w column y of the table.

5. a.

$\circ$	a	b	c	d
a	a	b	c	d
b	b	c	d	d
c	c	d	b	b
d	d	a	b	c

Notice that $d \circ b = a$, but $b \circ d \neq a$. So b and d have one-sided inverses but not inverses (two-sided).

c.

$\circ$	a	b	c	d
a	a	a	a	a
b	a	c	d	b
c	a	d	a	b
d	a	a	b	c

Notice that $(b \circ b) \circ c = c \circ c = a$ and $b \circ (b \circ c) = b \circ d = b$. Therefore, $\circ$ is not associative.

e.

$\circ$	a	b	c	d
a	a	b	c	d
b	b	a	a	a
c	c	a	a	a
d	d	a	a	a

Notice that $(b \circ b) \circ c = a \circ c = c$ and $b \circ (b \circ c) = b \circ a = b$. Therefore, $\circ$ is not associative.

6. a.

$\circ$	a	b
a	a	b
b	b	a

c.

$\circ$	a	b
a	b	b
b	b	b

7. Suppose the elements of table T are numbers $1, \ldots, n$. Check the equation $T(i, T(j, k)) = T(T(i, j), k)$ for all values of i, j, and k between 1 and n.

8. a. $f(g(x)) = g(g(g(x))) = g(f(x))$.
c. For example, $a, f(a), g(a), f(g(a))$.

9. a. $y = y \circ e = y \circ (x \circ x^{-1}) = (y \circ x) \circ x^{-1} = (z \circ x) \circ x^{-1} = z \circ (x \circ x^{-1}) = z \circ e = z$.

Section 10.2

1. No. Notice that $A \cdot B \neq B \cdot A$ because $A - B \neq B - A$. Similarly, 0 is not an identity for $+$, and 1 is not an identity for $\cdot$.

2. a. $x + x\,y = x(1 + y) = x\,1 = x$.

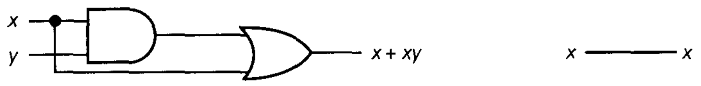
c. $x + \overline{x}y = (x + \overline{x})(x + y) = 1(x + y) = x + y.$

3. $\overline{e} = \overline{\overline{y}z + y\overline{z}} = \overline{\overline{y}z}\,\overline{y\overline{z}} = (y + \overline{z})(\overline{y} + z) = \overline{y}\,\overline{z} + yz.$

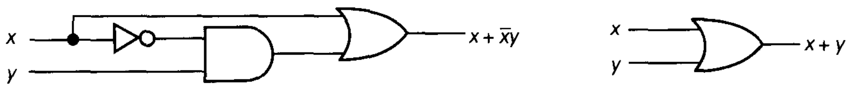
4. a. $\overline{x} + \overline{y} + xyz = \overline{xy} + (xy)z = \overline{xy} + z = \overline{x} + \overline{y} + z.$

5. a. $x + y.$
c. $\overline{y}(x + z).$
e. $x\,y + z.$
g. $x + y.$
i. 1.
k. $x + y.$

6. a.



c.



7. a. $x0.$
c. $(x + y)(x + z).$
e. $y(\overline{x} + z).$

9. Show that $\overline{a} + \overline{b}$ acts like the complement of ab. In other words, show that

$$(ab) + (\overline{a} + \overline{b}) = 1 \text{ and } (ab)(\overline{a} + \overline{b}) = 0.$$

The result then follows from (10.8). For the first equation we have

$$(ab) + (\overline{a} + \overline{b}) = (a + \overline{a} + \overline{b})(b + \overline{a} + \overline{b}) = (1 + \overline{b})(1 + \overline{a}) = (1)(1) = 1.$$

For the second equation we have

$$(ab)(\overline{a} + \overline{b}) = ab\overline{a} + ab\overline{b} = 0 + 0 = 0.$$

11. a. Since $x = xx$, we have $x \preceq x$. So $\preceq$ is reflexive. If $x \preceq y$ and $y \preceq x$, then $x = xy$ and $y = yx$. Therefore, $x = xy = yx = y$. Thus $\preceq$ is antisymmetric. If $x \preceq y$ and $y \preceq z$, then $x = xy$ and $y = yz$. Therefore, $x = xy = x(yz) = (xy)z = xz$. So $x \preceq z$. Thus $\preceq$ is transitive.

13. Since p occurs more than once in the factorization of n, it follows that n/p still contains at least one factor of p. For example, if $n = p^2q$, then $n/p = pq$. So $\text{lcm}(p, n/p) = n/p$, which is not equal to n (the unit of the algebra). Similarly, $\gcd(p, n/p) = p$, which is not 1 (the zero of the algebra). So properties of part 3 of the definition of a Boolean algebra fail to hold.

Section 10.3

1. monus(x, 0) = x, monus(0, y) = 0, monus($s(x)$, $s(y)$) = monus(x, y), where $s(x)$ denotes the successor of x.

3. reverse(L) = if isEmptyL(L) then L
else cat(reverse(tail(L)), ⟨head(L)⟩).

5. Let genlists(A) denote the set of general lists over A. The operations for general lists are similar to those for lists. The main difference is that the cons function and the head function have the following types to reflect the general nature of elements in a list.

$$\text{cons: } A \cup \text{genlists}(A) \times \text{genlists}(A) \to \text{genlists}(A),$$
$$\text{head: genlists}(A) \to \text{genlists}(A) \cup A.$$

The axioms are identical to those for lists.

7. post(⟨4, 5, −, 2, +⟩, ⟨ ⟩) = post(⟨5, −, 2, +⟩, ⟨4⟩) = post(⟨−, 2, +⟩, ⟨5, 4⟩) = post(⟨2, +⟩, eval(−, ⟨5, 4⟩)) = post(⟨2, +⟩, ⟨−1⟩) = post(⟨+⟩, ⟨2, −1⟩) = post(⟨ ⟩, eval(+, ⟨2, −1⟩)) = post(⟨ ⟩, ⟨1⟩) = 1.

9. In equational form we have preorder(emptyTree) = emptyQ, and preorder(tree(L, x, R)) = apQ(addQ(x, emptyQ), apQ(preorder(L), preorder(R))).

11. Remove (all occurrences of an element from a stack).

13. Let D be the set of deques over the set A. Then the carriers should be A, D, and Boolean. The operators can be defined as

$$\begin{aligned}
&\text{emptyD} \in D,\\
&\text{isEmptyD: } D \to \text{Boolean},\\
&\text{addLeft: } A \times D \to D,\\
&\text{addRight: } D \times A \to D,\\
&\text{left: } D \to A,\\
&\text{right: } D \to A,\\
&\text{deleLeft: } D \to D,\\
&\text{deleRight: } D \to D.
\end{aligned}$$

With axioms:

$$\begin{aligned}
&\text{isEmptyD(emptyD)} = \text{true},\\
&\text{isEmptyD(addLeft}(a, d)) = \text{isEmptyD(addRight}(d, a)) = \text{false},\\
&\text{left(addLeft}(a, d)) = \text{right(addRight}(d, a)) = a,\\
&\text{left(addRight}(d, a)) = \text{if isEmptyD}(d) \text{ then } a\\
&\qquad\qquad\qquad\qquad\quad \text{else left}(d),\\
&\text{right(addLeft}(a, d)) = \text{if isEmptyD}(d) \text{ then } a\\
&\qquad\qquad\qquad\qquad\quad \text{else right}(d),\\
&\text{deleLeft(addLeft}(a, d)) = \text{deleRight(addRight}(d, a)) = d,\\
&\text{deleLeft(addRight}(d, a)) = \text{if isEmptyD}(d) \text{ then emptyD}\\
&\qquad\qquad\qquad\qquad\qquad \text{else addRight(deleLeft}(d), a),\\
&\text{deleRight(addLeft}(a, d)) = \text{if isEmptyD}(d) \text{ then emptyD}\\
&\qquad\qquad\qquad\qquad\qquad \text{else addLeft}(a, \text{deleRight}(d)).
\end{aligned}$$

15. Let
$$\begin{aligned}
Q[A] &= D[A],\\
\text{emptyQ} &= \text{emptyD},\\
\text{isEmptyQ} &= \text{isEmptyD},\\
\text{frontQ} &= \text{left},\\
\text{deleQ} &= \text{deleLeft},\\
\text{addQ}(a, q) &= \text{addRight}(q, a).
\end{aligned}$$
Then the axioms are proved as follows:

$$\begin{aligned}
\text{isEmptyQ(emptyQ)} &= \text{isEmptyD(emptyD)} = \text{true}.\\
\text{isEmptyQ(addQ}(a, q)) &= \text{isEmptyD(addRight}(q, a)) = \text{false}.\\
\text{frontQ(addQ}(a, q)) &= \text{left(addRight}(q, a))\\
&= \text{if isEmptyD}(q) \text{ then } a \text{ else left}(q)\\
&= \text{if isEmptyQ}(q) \text{ then } a \text{ else frontQ}(q).\\
\text{delQ(addQ}(a, q)) &= \text{deleLeft(addRight}(q, a))\\
&= \text{if isEmptyD}(q) \text{ then emptyD}\\
&\qquad \text{else addRight(deleLeft}(q), a)\\
&= \text{if isEmptyQ}(q) \text{ then emptyQ}\\
&\qquad \text{else addQ}(a, \text{deleQ}(q)).
\end{aligned}$$

17. a. Let $P(x)$ denote the statement "$\text{plus}(x, s(y)) = s(\text{plus}(x, y))$ for all $y \in \mathbb{N}$." Certainly $P(0)$ is true because $\text{plus}(0, s(y)) = s(y) = s(\text{plus}(0, y))$. So assume that $P(x)$ is true, and prove that $P(s(x))$ is true. We can evaluate each expression in the statement of $P(s(x))$ as follows:

$$\begin{aligned}
\text{plus}(s(x), s(y)) &= s(\text{plus}(p(s(x)), s(y))) && \text{(by definition of plus)}\\
&= s(\text{plus}(x, s(y))) && (\text{since } p(s(x)) = x)\\
&= s(s(\text{plus}(x, y))) && \text{(by induction)},
\end{aligned}$$

and

$$\begin{aligned} s(\text{plus}(s(x),\, y)) &= s(s(\text{plus}(p(s(x)),\, y))) && \text{(by definition of plus)} \\ &= s(s(\text{plus}(x,\, y))) && \text{(since } p(s(x)) = x\text{).} \end{aligned}$$

Both expressions are equal. So $P(s(x))$ is true.

18. Induction will be with respect to the length of y. We'll use the notation $y : a$ for addQ(a, y). For the basis case we have the following equations, where $y =$ emptyQ:

apQ(x, emptyQ : a)

$$\begin{aligned} &= \text{apQ}(x : \text{front}(\text{emptyQ} : a),\, \text{delQ}(\text{emptyQ} : a)) && \text{(def. of apQ)} \\ &= \text{apQ}(x : a,\, \text{emptyQ}) && \text{(simplify)} \\ &= x : a && \text{(simplify)} \\ &= \text{apQ}(x : a,\, \text{emptyQ})) && \text{(def. of apQ).} \end{aligned}$$

For the induction case, assume that the equation is true for all queues y having length n, and show that the equation is true for the queue $y : b$, having length $n + 1$. Starting with the left side of the equation, we have

apQ(x, $y : b : a$)

$$\begin{aligned} &= \text{apQ}(x : \text{front}(y : b : \text{a}),\, \text{delQ}(y : b : a)) && \text{(def of apQ)} \\ &= \text{apQ}(x\text{:front}(y : b),\, \text{delQ}(y : b : a)) && (\text{front}(y : b : a) = \text{front}(y : b)) \\ &= \text{apQ}(x : \text{front}(y{:}b),\, \text{delQ}(y : b) : a) && (\text{delQ}(y : b : a) = \text{delQ}(y : b) : a) \\ &= \text{apQ}(x : \text{front}(y : b),\, \text{delQ}(y : b)) : a && \text{(induction)} \\ &= \text{apQ}(x,\, y : b) : a && \text{(def of apQ).} \end{aligned}$$

Section 10.4

1. a. $\{(1, a, \#, M), (2, a, *, N), (3, a, \%, N)\}$.
c. $\{(1, a, \#, M, x), (1, a, \#, M, z)\}$.
e. $\{(a, M), (b, M)\}$.

2. For example, if we let $R = \{(1, a), (2, b)\}$ and $S = \{(1, a)\}$, then join(R, S) $= \{(1, a)\}$ and $R \cup S = \{(1, a), (2, b)\}$

3. a. project(Channel, {Station, Cable}).
c. project(select(select(Rooms, Computer, Yes), BoardType, White), {Place}).
e. select(join(Channel, Program), Station, ESPN).

4. a. $t \in \text{select}_{A=a}(\text{select}_{B=b}(R))$ iff $t \in \text{select}_{B=b}(R)$ and $t(A) = a$ iff $t \in R$ and $t(B) = b$ and $t(A) = a$ iff $t \in \text{select}_{A=a}(R)$ and $t(B) = b$ iff $t \in \text{select}_{B=b}(\text{select}_{A=a}(R))$.
c. Let I, J, and K be the attribute sets for R, S, and T, respectively. Use the definition of join to show that $u \in (R \bowtie S) \bowtie T$ iff there exist $r \in R$ and $s \in S$ and $t \in$ T such that $u(a) = r(a)$ for all $a \in I$ and $u(a) = s(a)$ for all $a \in J$ and $u(a) = t(a)$ for all $a \in K$ iff $u \in R \bowtie (S \bowtie T)$.

5. $s \in \text{project}_X(\text{select}_{A=a}(R))$ iff there exists $t \in \text{select}_{A=a}(R)$ such that $s(B) = t(B)$ for all $B \in X$ iff there exists $t \in R$ such that $t(A) = a$ and $s(B) = t(B)$ for all $B \in X$ iff $s \in \text{project}_X(R)$ and $s(A) = a$ iff $s \in \text{select}_{A=a}(\text{project}_X(R))$.

6. **a.** Let f = seqPairs, where
seqPairs = eq0 $\rightarrow$ ~ ((0, 0)); apndr @ [seqPairs @ sub1, [id, id]].

7. **a.** For any pair of numbers (m, n), all three expressions compute the value of the expression $m + n$.

8. If c returns 0, then $* @ [a, g @ [b, c]] = * @ [a, b] = \text{g} @ [* @ [a, b], c]$, which proves the basis case. Now assume that c returns a positive number and (10.12) holds for sub1 @ c. We'll prove that (10.12) holds for c as follows, starting with the left side:

$$\begin{aligned} * @ [a, g @ [b, c]] &= * @ [a, (\text{eq0} @ 2 \rightarrow 1; g @ [*, sub1 @ 2]) @ [b, c]] && \text{(def of } g) \\ &= * @ [a, g @ [* @ [b, c], \text{sub1} @ c]] && (\text{eq0} @ c = \text{false}) \\ &= g @ [* @ [a, * @ [b, c]], \text{sub1} @ c] && \text{(induction).} \end{aligned}$$

Now look at the right side:

$$\begin{aligned} g @ [* @ [a, b], c] &= g @ [*, \text{sub1} @ 2] @ [* @ [a, b], c] && \text{(def of } g) \\ &= g @ [* @ [* @ [a, b], c], \text{ sub1} @ c]. \end{aligned}$$

It follows that the two sides are equal because multiplication is associative:

$$* @ [a, * @ [b, c]] = * @ [* @ [a, b], c].$$

Section 10.5

1.

+	[0]	[1]	[2]	[3]
[0]	[0]	[1]	[2]	[3]
[1]	[1]	[2]	[3]	[0]
[2]	[2]	[3]	[0]	[1]
[3]	[3]	[0]	[1]	[2]

·	[0]	[1]	[2]	[3]
[0]	[0]	[0]	[0]	[0]
[1]	[0]	[1]	[2]	[3]
[2]	[0]	[2]	[0]	[2]
[3]	[0]	[3]	[2]	[1]

.

2. **a.** $x = 99$. **c.** $x = 59$. **e.** 21.

3. Add multiples of n to x until the sum is positive. In other words, there is some k such that $x + kn > 0$. Set $y = x + kn$. It follows that $y \equiv x \pmod{n}$.

5. **a.** e = 11 works. **c.** e = 317 works.

7. **a.** Yes.
c. No. $4 +_9 6 = 1 \notin \{0, 2, 4, 6, 8\}$.

8. **a.** $\{0, 6\}$.
c. $\mathbb{N}_{12}$.

9. The three morphisms are f, g, and h, where: f is the zero function; $g(0) = 0$, $g(1) = 2$, $g(2) = 4$; $h(0) = 0$, $h(1) = 4$, $h(2) = 2$.

11. Notice that abs(1 + (–1)) = abs(0) = 0, but that abs(1) + abs(–1) = 1 + 1 = 2. So in general abs(x + y) $\neq$ abs(x) + abs(y).

13. **a.** $\{(ab)^n b \mid n \in \mathbb{N}\}$. **c.** $\varnothing$. **e.** $\{ba^n \mid n \in \mathbb{N}\}$.

Bibliography

In addition to the books and papers specifically referenced in this book, we also include some general references.

Andrews, P. B., *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof.* Academic Press, New York, 1986.

Appel, K., and W. Haken, Every planar map is four colorable. *Bulletin of the American Mathematical Society 82* (1976), 711–712.

Appel, K., and W. Haken, The solution of the four-color-map problem. *Scientific American 237* (1977), 108–121.

Apt, K. R., Ten years of Hoare's logic: A survey—Part 1. *ACM Transactions on Programming Languages and Systems 3* (1981), 431–483.

Backus, J., Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *Communications of the ACM 21* (1978), 613–641.

Chang, C., and R. C. Lee, *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York, 1973.

Cichelli, R. J., Minimal perfect hash functions made simple. *Communications of the ACM 23* (1980), 17–19.

Coppersmith, D., and S. Winograd, Matrix multiplication via arithmetic progressions. *Proceedings of 19th Annual ACM Symposium on the Theory of Computing* (1987), 1–6.

Delong, H., *A Profile of Mathematical Logic.* Addison-Wesley, Reading, MA, 1970.

Floyd, R. W., Algorithm 97: Shortest path. *Communications of the ACM 5* (1962), 345.

Floyd, R. W., Assigning meanings to programs. *Proceedings AMS Symposium Applied Mathematics, 19,* AMS, Providence, RI, 1967, pp. 19–31.

Frege, G., *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens.* Halle, 1879.

Galler, B. A., and M. J. Fischer, An improved equivalence algorithm. *Communications of the ACM 7* (1964), 301–303.

Gentzen, G., Untersuchungen uber das logische Schliessen. *Mathematische Zeitschrift 39* (1935), 176–210, 405–431; English translation: Investigation into logical deduction, *The Collected Papers of Gerhard Gentzen*, ed. M. E. Szabo. North-Holland, Amsterdam, 1969, pp. 68–131.

Gödel, K., Die Vollständigkeit der Axiome des logischen Funktionenkalküls. *Monatshefte für Mathematic und Physik 37* (1930), 349–360.

Gödel, K., Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematic und Physik 38* (1931), 173–198.

Graham, R. L., D. E. Knuth, and O. Patashnik, *Concrete Mathematics.* Addison-Wesley, Reading, MA, 1989.

Halmos, P. R., *Naive Set Theory.* Van Nostrand, New York, 1960.

Hamilton, A. G., *Logic for Mathematicians.* Cambridge University Press, New York, 1978.

Hilbert, D., and W. Ackermann, *Principles of Mathematical Logic.* (1938). Translated by Lewis M. Hammond, George G. Leckie, and F. Steinhardt. Edited by Robert E. Luce. Chelsea, New York, 1950.

Hoare, C.A.R., An axiomatic basis for computer programming. *Communications of the ACM 12* (1969), 576–583.

Kleene, S. C., *Introduction to Metamathematics.* Van Nostrand, New York, 1952.

Kleene, S. C., *Mathematical Logic.* John Wiley, New York, 1967.

Knuth, D. E., On the translation of languages from left to right. *Information and Control 8* (1965), 607–639.

Knuth, D. E., Two notes on notation. *The American Mathematical Monthly 99* (1992), 403–422.

Kruskal, J. B., Jr., On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society 7* (1956), 48–50.

Lukasiewicz, J., *Elementary Logiki Matematycznej.* PWN (Polish Scientific Publishers), 1929; translated as *Elements of Mathematical Logic*, Pergamon, Elmsford, NY, 1963.

Mallows, C. L., Conway's challenge sequence. *The American Mathematical Monthly 98* (1991), 5–20.

Martelli, A., and U. Montanari, An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems 4* (1982), 258–282.

Mendelson, E., *Introduction to Mathematical Logic.* Van Nostrand, New York, 1964.

Nagel, E., and J. R. Newman, *Gödel's Proof.* New York University Press, New York, 1958.

Pan, V., Strassen's algorithm is not optimal. *Proceedings of 19th Annual IEEE Symposium on the Foundations of Computer Science* (1978), 166–176.

Paterson, M. S., and M. N. Wegman, Linear Unification. *Journal of Computer and Systems Sciences 16* (1978), 158–167.

Paulson, L. C., *Logic and Computation.* Cambridge University Press, New York, 1987.

Prim, R. C., Shortest connection networks and some generalizations. *Bell System Technical Journal 36* (1957), 1389–1401.

Rivest, R. L., A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21* (1978), 120–126.

Robinson, J. A., A machine-oriented logic based on the resolution principle. *Journal of the ACM 12* (1965), 23–41.

Schöning, U., *Logic for Computer Scientists.* Birkhauser, Boston, 1989.

Skolem, T., Uber de mathematische logik. *Norsk Matematisk Tidsskrift 10* (1928), 125–142. Translated in ed. Jean van Heijenoort. *From Frege to Godel: A Source Book in Mathematical Logic 1879–1931*, Harvard University Press, Cambridge, MA, 1967, pp. 508–524.

Snyder, W., and J. Gallier, Higher-order unification revisited: Complete sets of transformations. *Journal of Symbolic Computation 8* (1989), 101–140.

Stanat, D. F., and D. F. McAllister, *Discrete Mathematics in Computer Science.* Prentice-Hall, Englewood Cliffs, NJ, 1977.

Strassen, V., Gaussian elimination is not optimal. *Numerische Mathematik 13* (1969), 354–356.

Suppes, P., *Introduction to Logic.* Van Nostrand, New York, 1957.

Warren, D. S., Memoing for logic programs. *Communications of the ACM 35* (1992), 93–111.

Warshall, S., A theorem on Boolean matrices, *Journal of the ACM* 9 (1962), 11–12.

Whitehead, A. N., and B. Russell, *Principia Mathematica.* Cambridge University Press, New York, 1910.

Wos, L., R. Overbeek, E. Lusk, and J. Boyle, *Automated Reasoning: Introduction and Applications.* Prentice-Hall, Englewood Cliffs, NJ, 1984.

Greek Alphabet

A	α	alpha
B	β	beta
Γ	γ	gamma
Δ	δ	delta
E	ϵ	epsilon
Z	ζ	zeta
H	η	eta
Θ	θ	theta
I	ι	iota
K	κ	kappa
Λ	λ	lambda
M	μ	mu
N	ν	nu
Ξ	ξ	xi
O	o	omicron
Π	π	pi
P	ρ	rho
Σ	σ	sigma
T	τ	tau
Υ	υ	upsilon
Φ	ϕ	phi
X	χ	chi
Ψ	ψ	psi
Ω	ω	omega

Greek Alphabet

A	α	alpha
B	β	beta
Γ	γ	gamma
Δ	δ	delta
E	ϵ	epsilon
Z	ζ	zeta
H	η	eta
Θ	θ	theta
I	ι	iota
K	κ	kappa
Λ	λ	lambda
M	μ	mu
N	ν	nu
Ξ	ξ	xi
O	o	omicron
Π	π	pi
P	ρ	rho
Σ	σ	sigma
T	τ	tau
Υ	υ	upsilon
Φ	ϕ	phi
X	χ	chi
Ψ	ψ	psi
Ω	ω	omega

Symbol Glossary

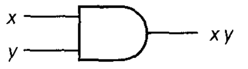
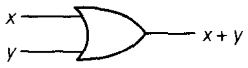
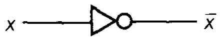
Each symbol or expression is listed with a short definition and the page number where it first occurs. The list is ordered by page number.

$d \mid n$	d divides n with no remainder 6
$x \in S$	x is an element of S 13
$x \notin S$	x is not an element of S 13
$\ldots$	ellipsis 14
$\varnothing$	the empty set 14
$\mathbb{N}$	natural numbers 15
$\mathbb{Z}$	integers 15
$\mathbb{Q}$	rational numbers 15
$\mathbb{R}$	real numbers 15
$\{x \mid P\}$	set of all x satisfying property P 16
$A \subset B$	A is a subset of B 16
$A \not\subset B$	A is not a subset of B 16
$A \cup B$	A union B 19
$A \cap B$	A intersection B 21
$A - B$	difference: elements in A but not B 22
$A \oplus B$	symmetric difference: $(A - B) \cup (B - A)$ 23
A'	complement of A 23

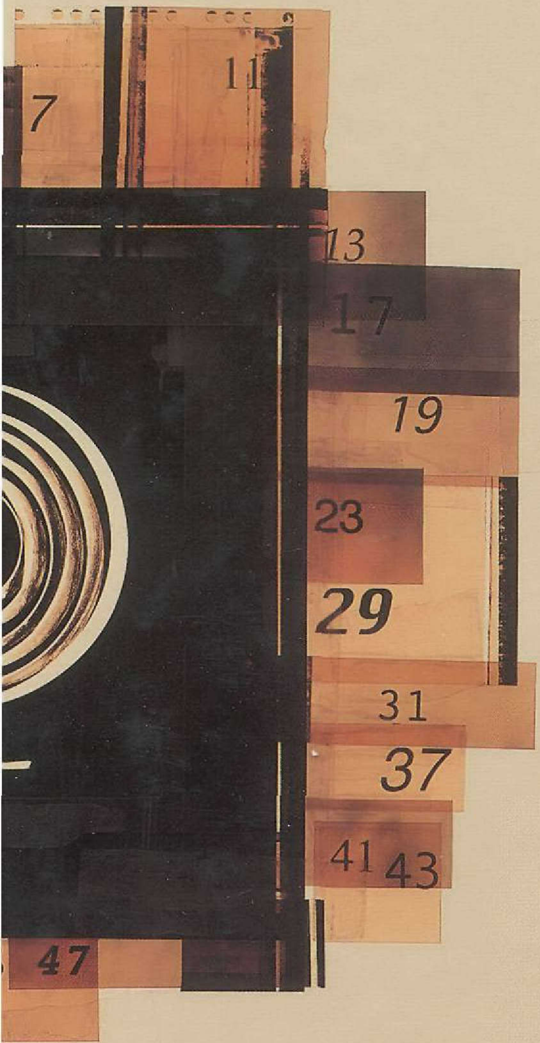
$\|A\|$	cardinality of A 26
$[a, b, b, a]$	bag, or multiset, of four elements 29
(x, y, x)	tuple of three elements 35
$(\,)$	empty tuple 36
$A \times B$	Cartesian product $\{(a, b) \mid a \in A \text{ and } b \in B\}$ 36
$\langle x, y, x\rangle$	list of three elements 39
$\langle\,\rangle$	empty list 39
$\text{cons}(x, t)$	list with head x and tail t 40
$\text{lists}(A)$	set of all lists over A 40
Λ	empty string 41
$\|s\|$	length of string s 41
A^*	set of all strings over alphabet A 42
LM	product of languages L and M 43
L^n	product of language L with itself n times 44
L^*	closure of language L 44
L^+	positive closure of language L 44
$R(a, b, c)$	(a, b, c) is in the relation R 47
$x\ R\ y$	$R(x, y)$ or x is related by R to y 47
$f : A \to B$	function type: f has domain A and codomain B 74
$f(C)$	image of C under f 76
$f^{-1}(D)$	pre-image of D under f 76
$\lfloor x\rfloor$	floor of x: largest integer $\leq x$ 79
$\lceil x\rceil$	ceiling of x: smallest integer $\geq x$ 79
$\text{gcd}(a, b)$	greatest common divisor of a and b 80
$a \bmod b$	remainder upon division of a by b 82
$\mathbb{N}_n$	the set $\{0, 1, \ldots, n - 1\}$ 83
χ_B	characteristic function for subset B 89

$f \circ g$	composition of functions f and g 91
f^{-1}	inverse of bijective function f 103
$x :: t$	list with head x and tail t 135
$\text{tree}(L, x, R)$	binary tree with root x and subtrees L and R 138
$\Sigma\ a_i$	sum of the numbers a_i 150
$\Pi\ a_i$	product of the numbers a_i 150
$n!$	n factorial: $n \cdot (n-1) \cdots 1$ 150
$A \to \alpha$	grammar production 174
$A \to \alpha \mid \beta$	grammar productions $A \to \alpha$ and $A \to \beta$ 177
$A \Rightarrow \alpha$	A derives α in one step 178
$A \Rightarrow^+ \alpha$	A derives α in one or more steps 178
$A \Rightarrow^* \alpha$	A derives α in zero or more steps 178
$L(G)$	language of grammar G 178
$R \circ S$	composition of binary relations R and S 195
$r(R)$	reflexive closure of R 199
R^c	converse of relation R 200
$s(R)$	symmetric closure of R 200
$t(R)$	transitive closure of R 200
R^+	transitive closure of R 203
R^*	reflexive transitive closure of R 203
$[x]$	equivalence class of things equivalent to x 218
$\text{tsr}(R)$	smallest equivalence relation containing R 225
$\langle A, \prec \rangle$	irreflexive partially ordered set 235
$\langle A, \preceq \rangle$	reflexive partially ordered set 235
$x \preceq y$	$x \prec y$ or $x = y$ 235
$x \prec y$	x is less than y or x is a predecessor of y 236
W_A	worst-case function for algorithm A 275

$P(n, r)$	number of permutations of n things taken r at a time 290
$C(n, r)$	number of combinations of n things taken r at a time 294
$\binom{n}{r}$	binomial coefficient symbol 294
$P(A)$	probability of event A 300
$\Theta(f)$	big theta: same growth rate as f 334
$o(f)$	little oh: lower growth rate than f 338
$O(f)$	big oh: growth rate bounded above by that of f 339
$\Omega(f)$	big omega: growth rate bounded below by that of f 340
$\neg P$	logical negation of P 349
$P \wedge Q$	logical conjunction of P and Q 349
$P \vee Q$	logical disjunction of P and Q 349
$P \rightarrow Q$	logical conditional: P implies Q 349
$P \equiv Q$	logical equivalence of P and Q 349
$\therefore$	therefore 370
$\vdash W$	turnstile to denote W is a theorem 394
$\exists x$	existential quantifier: there is an x 399
$\forall x$	universal quantifier: for all x 399
$W(x/t)$	wff obtained from W by replacing free x's by t 405
x/t	binding of the variable x to the term t 405
$W(x)$	W contains a free variable x 407
$\{P\}\ S\ \{Q\}$	S has precondition P and postcondition Q 467
$\square$	empty clause: a contradiction 506
$\{x/t,\ y/s\}$	substitution containing two bindings 514
ϵ	empty substitution 515
$E\theta$	instance of E: substitution θ applied to E 515
$\theta\sigma$	composition of substitutions θ and σ 515
$C\theta - N$	remove all occurrences of N from clause $C\theta$ 522

$R(S)$	resolution of clauses in the set S 524
$C \leftarrow A,\ B$	logic program clause: C if A and B 536
$\leftarrow A$	logic program goal: is A true? 537
$\langle A; s, a\rangle$	algebra with carrier A and operations s and a 561
$\overline{x}$	complement of Boolean algebra variable x 573
	AND gate 579
	OR gate 579
	NOT gate 579
$R \bowtie S$	join of relations R and S 606
$x \equiv y \pmod{n}$	congruence mod n: x mod $n = y$ mod n 614

Index



Discrete Mathematics

SECOND EDITION

James L. Hein

This introduction to discrete mathematics prepares future computer scientists engineers, and mathematicians for success by providing extensive and concentrated coverage of logic, functions, algorithmic analysis, and algebraic structures. *Discrete Mathematics, Second Edition* illustrates the relationships between key concepts through its thematic organization and provides a seamless transition between subjects. Distinct for the depth with which it covers logic, this text emphasizes problem solving and the application of theory as it carefully guides the reader from basic to more complex topics. *Discrete Mathematics* is an ideal resource for discovering the fundamentals of discrete math.

Features

- Includes over 1,500 exercises, with proofs and challenges. Answers are provided for over half of the exercises.
- Each section contains updated material, and the addition of new subject headings makes it easy to identify topics.
- Algorithms are presented in a variety of ways to accommodate a multitude of learning styles.

Jones and Bartlett Publishers
40 Tall Pine Drive
Sudbury, MA 01776
978-443-5000
info@jbpub.com
http://math.jbpub.com/

ISBN 0-7637-2210-3
90000
9 780763 722104