Formal Languages and Computation

Models and Their Applications

Alexander Meduna

CRC Press
Taylor & Francis Group
AN AUERBACH BOOK

Formal Languages and Computation

Models and Their Applications

Formal Languages and Computation

Models and Their Applications

Alexander Meduna

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
AN AUERBACH BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

© 2014 by Taylor & Francis Group, LLC
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works
Version Date: 20140106

International Standard Book Number-13: 978-1-4665-1349-5 (eBook - PDF)

This book contains information obtained from authentic and highly regarded sources. Reasonable efforts have been made to publish reliable data and information, but the author and publisher cannot assume responsibility for the validity of all materials or the consequences of their use. The authors and publishers have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission to publish in this form has not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

Except as permitted under U.S. Copyright Law, no part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, microfilming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, please access www.copyright.com (http://www.copyright.com/) or contact the Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC is a not-for-profit organization that provides licenses and registration for a variety of users. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

To Ivana and Zbyněk and
in memory of Meister Eckhart and Gustav Mahler

Contents

Preface

This book is designed to serve as a text for a one-semester introductory course in the theory of formal languages and computation. It covers all the traditional topics of this theory, such as automata, grammars, parsing, computability, decidability, computational complexity, and properties of formal languages. Special attention is paid to the fundamental models for formal languages and their applications in computer science.

Subject

Formal language theory defines languages mathematically as sets of sequences consisting of symbols. This definition encompasses almost all languages as they are commonly understood. Indeed, natural languages, such as English, are included in this definition. Of course, all artificial languages introduced by various scientific disciplines can be viewed as formal languages; perhaps most illustratively, every programming language represents a formal language in terms of this definition. It thus comes as no surprise that formal language theory, which represents a mathematically systematized body of knowledge concerning formal languages, is important to all the scientific areas that make use of these languages to a certain extent.

The theory of formal languages represents the principal subject of this book. The text focuses its attention on the fundamental models for formal languages and their computation-related applications in computer science, hence its title *Formal Languages and Computation*: *Models and Their Applications.*

Models

The strictly mathematical approach to languages necessitates introducing formal models that define them. Most models of this kind are underlain by *rewriting systems*, which are based on rules by which they repeatedly change sequences of symbols, called strings. Despite their broad variety, most of them can be classified into two basic categories—generative and accepting language models. Generative models, better known as *grammars*, define strings of their language, so their rewriting process generates them from a special start symbol. On the other hand, accepting models, better known as *automata*, define strings of their language by a rewriting process that starts from these strings and ends in a prescribed set of final strings.

Applications

The book presents applications of language models in both practical and theoretical computer science.

In practice, the text explains how appropriate language models underlie computer science engineering techniques used in *language processors*. It pays special attention to *programming language analyzers* based on four language models—regular expressions, finite automata, context-free grammars, and pushdown automata. More specifically, by using *regular expressions* and *finite automata*, it builds up *lexical analyzers*, which recognize lexical units and verify that they are properly formed. Based on *context-free grammars* and *pushdown automata*, it creates *syntax analyzers*, which recognize syntactic structures in computer programs and verify that they are correctly written according to grammatical rules. That is, the text first explains how to specify the programming language syntax by using context-free grammars, which are considered the most widely used specification tool for this purpose. Then, it describes how to write syntax analyzers based on pushdown automata.

In theory, the book makes use of language-defining models to explore the very heart of the *foundations of computation*. That is, the text introduces the mathematical notion of a *Turing machine*, which has become a universally accepted formalization of the intuitive notion of a procedure. Based on this strictly mathematical notion, it studies the general limits of computation. More specifically, it performs this study in terms of two important topics concerning computation—computability and decidability. Regarding *computability*, it considers Turing machines as computers of functions over nonnegative integers and demonstrates the existence of functions whose computation cannot be specified by any procedure. As far as *decidability* is concerned, it formalizes problem-deciding algorithms by Turing machines that halt on every input. The book formulates several important problems concerning the language models discussed earlier in this book and constructs algorithms that decide them. On the other hand, it describes several problems that are not decidable by any algorithm. Apart from giving several specific undecidable problems, this book builds up a general theory of undecidability. Finally, the text approaches decidability in a much finer and realistic way. Indeed, it reconsiders problem-deciding algorithms in terms of their computational complexity measured according to time and space requirements. Perhaps most importantly, it shows that although some problems are decidable in principle, they are intractable for absurdly high computational requirements of the algorithms that decide them.

Use

As already stated, this book is intended as a textbook for a one-term introductory course in formal language theory and its applications in computer science.

Second, the book can also be used as an accompanying textbook for a compiler class at an undergraduate level because the text allows the flexibility needed to select only the topics relevant to compilers.

Finally, this book is useful to all researchers, including people out of computer science, who somehow deal with formal languages and their models in their scientific fields.

Approach

Primarily, this book represents a theoretically oriented treatment of formal languages and their models. Indeed, it introduces all formalisms concerning them with enough rigor to make all results quite clear and valid. Every complicated mathematical passage is preceded

by its intuitive explanation so that even the most complex parts of the book are easy to grasp. Every new concept or algorithm is preceded by an explanation of its purpose and followed by some examples with comments to reinforce its understanding. All applications are given in a quite realistic way to clearly demonstrate a strong relation between the theoretical concepts and their uses.

Secondarily, as already pointed out, the text also presents several significant applications of formal languages and their models in practice. All applications are given in a quite realistic way to clearly show a close relation between the theoretical concepts and their uses.

Prerequisites

On the part of the student, no previous knowledge concerning formal languages is assumed. Although this book is self-contained, in the sense that no other sources are needed for understanding the material, a familiarity with the rudiments of discrete mathematics is helpful for a quick comprehension of formal language theory. A familiarity with a high-level programming language helps to grasp the material concerning applications in this book.

Organization

Synopsis

The entire text contains 12 chapters, which are divided into 5 sections.

Section I, which consists of Chapters 1 and 2, gives an introduction to the subject. Chapter 1 recalls the basic mathematical notions used in the book. Chapter 2 gives the basics of formal languages and rewriting systems that define them.

Section II, which consists of Chapters 3 through 5, studies regular languages and their models. Chapter 3 gives the basic definitions of these languages and their models, such as regular expressions and finite automata. Chapter 4 is application oriented; specifically, it builds lexical analyzers by using models for regular languages. Chapter 5 studies properties concerning regular languages.

Section III, which consists of Chapters 6 through 8, discusses context-free languages and their models. To a large extent, its structure parallels Section II. Indeed, Chapter 6 defines context-free languages and their models, including context-free grammars and pushdown automata. Chapter 7 explains how to construct syntax analyzers based on these grammars and automata. Chapter 8 establishes certain properties concerning context-free languages.

Section IV, which consists of Chapters 9 through 11, concerns Turing machines as a formalization of algorithms. Chapter 9 defines them. Based on Turing machines, Chapter 10 gives the basic ideas, concepts, and results underlying the theory of computation and its crucially important parts, including computability, decidability, and computational complexity. Simultaneously, this chapter establishes important properties concerning languages defined by Turing machines. Chapter 11 presents the essentials concerning general grammars, which represent grammatical counterparts to Turing machines.

Section V consists of Chapter 12. This chapter summarizes the entire textbook, points out selected modern trends, makes many historical and bibliographical remarks, and recommends further reading to the serious student.

Finally, the book contains two appendices. Appendix I gives the index to mathematical symbols used in the text. Appendix II contains the alphabetic index that lists all important language models introduced in the book.

Numbering

Regarding the technical organization of the text, algorithms, conventions, corollaries, definitions, lemmas, and theorems are sequentially numbered within chapters. Examples and figures are organized similarly. The end of conventions, corollaries, definitions, lemmas, and theorems is denoted by ■.

Exercises

At the end of each chapter, a set of exercises is given to reinforce and augment the material covered. Selected exercises, denoted by S, have their solutions or parts of them at the end of the chapter.

Algorithms

This textbook contains many algorithms. Strictly speaking, every algorithm requires a verification that it terminates and works correctly. However, the termination of the algorithms given in this book is always so obvious that its verification is omitted throughout. The correctness of complicated algorithms is verified in detail. On the other hand, we most often give only the gist of the straightforward algorithms and leave their rigorous verification as an exercise. The text describes the algorithms in Pascal-like notation, which is so simple and intuitive that even the student unfamiliar with the Pascal programming language can immediately pick it up. In this description, a Pascal **repeat** loop is sometimes ended with **until no change**, meaning that the loop is repeated until no change can result from its further repetition. As the clear comprehensibility is a paramount importance in the book, the description of algorithms is often enriched by an explanation in words.

Support on the World Wide Web

Further backup materials, including lecture notes, are available at http://www.fit.vutbr.cz/~meduna/books/flc.

Acknowledgments

For almost a decade, I taught the theory of formal languages and computation at the University of Missouri-Columbia in the United States back in the 1990s, and since 2000, I have taught this subject at the Brno University of Technology in the Czech Republic. The lecture notes I wrote at these two universities underlie this book, and I have greatly benefited from conversations with many colleagues and students there. In addition, this book is based on notes I have used for my talks at various American, Asian, and European universities over the past three decades. Notes made at the Kyoto Sangyo University in Japan were particularly helpful.

Writing this book was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070). This work was also supported by the Visual Computing Competence Center (TE01010415).

My thanks to Martin Čermák and Jiří Techet for their comments on a draft of this text. Without the great collaboration, encouragement, and friendship with Zbyněk Křivka, I would have hardly started writing this book, let alone complete it. I am also grateful to John Wyzalek at Taylor & Francis for excellent editorial work. Finally, I thank my wife Ivana for her patience and, most importantly, love.

Author

Alexander Meduna, PhD, is a full professor of computer science at the Brno University of Technology in the Czech Republic, where he earned his doctorate in 1988. From 1988 until 1997, he taught computer science at the University of Missouri-Columbia in the United States. Even more intensively, since 2000, he has taught computer science and mathematics at the Brno University of Technology. In addition to these two universities, he has taught computer science at several other American, European, and Japanese universities for shorter periods of time. His classes have been primarily focused on formal language theory and its applications in theoretical and practical computer science. His teaching has also covered various topics including automata, discrete mathematics, operating systems, and principles of programming languages. Among many other awards for his scholarship and writing, he received the Distinguished University Professor Award from Siemens in 2012. He very much enjoys teaching classes related to the subject of this book.

Dr. Meduna has written several books. Specifically, he is the author of two textbooks—*Automata and Languages* (Springer, 2000) and *Elements of Compiler Design* (Taylor & Francis, 2008; translated into Chinese in 2009). Furthermore, he is the coauthor of three monographs—*Grammars with Context Conditions and Their Applications* (along with Martin Švec, Wiley, 2005), *Scattered Context Grammars and Their Applications* (with Jiří Techet, WIT Press, 2010), and *Regulated Grammars and Automata* (with Petr Zemek, Springer, 2014). He has published over 90 studies in prominent international journals, such as *Acta Informatica* (Springer), *International Journal of Computer Mathematics* (Taylor & Francis), and *Theoretical Computer Science* (Elsevier). All his scientific work discusses the theory of formal languages and computation, the subject of this book, or closely related topics, such as compiler writing.

Alexander Meduna's website is http://www.fit.vutbr.cz/~meduna. His scientific work is described in detail at http://www.fit.vutbr.cz/~meduna/work.

INTRODUCTION I

In this two-chapter introductory section, we first review the mathematical notions, concepts, and techniques used throughout this book to express all the upcoming theory of formal languages clearly and precisely. Then, we introduce formal languages defined by rewriting systems, and we also explain how these systems underlie important models used in both practical and theoretical computer science. We conclude this section by giving a synopsis of the entire book.

Chapter 1 reviews the principal ideas and notions underlying some mathematical areas because they are needed for understanding this book. These areas include logic, set theory, discrete mathematics, and graph theory.

Chapter 2 gives an introduction to this work. It defines its two central notions—formal languages and rewriting systems—and demonstrates how to use them as language-defining models and models of computation. In terms of these models, this chapter closes its discussion by presenting a synopsis of the entire book.

Chapter 1

Mathematical Background

This chapter reviews rudimentary concepts from logic (Section 1.1), set theory (Section 1.2), discrete mathematics (Section 1.3), and graph theory (Section 1.4). For readers familiar with them, this chapter can be skipped and treated as a reference for notation and definitions.

1.1 Logic

In this section, we review the basics of elementary logic. We pay a special attention to the fundamental proof techniques used in this book.

In general, a *formal mathematical system S* consists of basic *symbols*, *formation rules*, *axioms*, and *inference rules*. Basic symbols, such as constants and operators, form components of *statements* that are composed according to formation rules. Axioms are primitive statements, whose validity is accepted without justification. By inference rules, some statements infer other statements. A *proof* of a statement s in S consists of a sequence of statements $s_1, \ldots, s_i, \ldots, s_n$ such that $s = s_n$ and each s_i is either an axiom of S or a statement inferred by some of the statements $s_1, \ldots, s_{i-1}$ according to the inference rules; s proved in this way represents a *theorem* of S.

Logical connectives join statements to create more-complicated statements. The most common logical connectives are *not*, *and*, *or*, *implies*, and *if and only if*. In this list, *not* is unary while the other connectives are binary. That is, if s is a statement, then *not* s is a statement as well. Similarly, if s_1 and s_2 are statements, then s_1 *and* s_2, s_1 *or* s_2, s_1 *implies* s_2, and s_1 *if and only if* s_2 are statements, too. We often write $\neg$, $\wedge$, and $\vee$ instead of *not*, *and*, and *or*, respectively. The following *truth table* presents the rules governing the *truth* or *falsity* concerning statements connected by the binary connectives. Regarding the unary connective $\neg$, if s is true, then $\neg s$ is false, and if s is false, then $\neg s$ is true.

Convention 1.1 Throughout this book, *truth* and *falsity* are denoted by **1** and **0**, respectively. ■

By the truth table (Figure 1.1), s_1 *and* s_2 is true if both statements are true; otherwise, s_1 *and* s_2 is false. Analogically, we can interpret the other rules governing the truth or falsity of a statement containing the other connectives from this table. A statement of *equivalence*, which has the form

s_1	s_2	$\wedge$	$\vee$	*implies*	*if and only if*
0	**0**	**0**	**0**	**1**	**1**
0	**1**	**0**	**1**	**1**	**0**
1	**0**	**0**	**1**	**0**	**0**
1	**1**	**1**	**1**	**1**	**1**

Figure 1.1 Truth table.

s_1 *if and only if* s_2, sometimes abbreviated to s_1 *iff* s_2, plays a crucial role in this book. A proof that it is true usually consists of two parts. The *only-if* part demonstrates that s_1 *implies* s_2 is true although the *if* part proves that s_2 *implies* s_1 is true.

Example 1.1 There exists a useful way of representing ordinary *infix arithmetic expressions* without using parentheses. This notation is referred to as *Polish notation* that has two fundamental forms—*postfix* and *prefix notation*. The former is defined recursively as follows:

Let Ω be a set of binary operators, and let Σ be a set of operands.

1. Every $a \in \Sigma$ is a postfix representation of a.
2. Let AoB be an infix expression, where $o \in \Omega$, and A, B are infix expressions. Then, CDo is the postfix representation of AoB, where C and D are the postfix representations of A and B, respectively.
3. Let C be the postfix representation of an infix expression A. Then, C is the postfix representation of (A).

Consider the infix logical expression $(\mathbf{1} \vee \mathbf{0}) \wedge \mathbf{0}$. The postfix expressions for **1** and **0** are **1** and **0**, respectively. The postfix expression for $\mathbf{1} \vee \mathbf{0}$ is $\mathbf{1}\ \mathbf{0} \vee$, so the postfix expression for $(\mathbf{1} \vee \mathbf{0})$ is $\mathbf{1}\ \mathbf{0} \vee$, too. Thus the postfix expression for $(\mathbf{1} \vee \mathbf{0}) \wedge \mathbf{0}$ is $\mathbf{1}\ \mathbf{0} \vee \mathbf{0} \wedge$.

The prefix notation is defined analogically, except that in the second part of the definition, o is placed in front of AB; the details are left as an exercise.

There exist many logic laws useful to demonstrate that an implication is true. Specifically, the *contrapositive law* says $(s_1$ *implies* $s_2)$ *if and only if* $((\neg s_2)$ *implies* $(\neg s_1))$, so we can prove s_1 *implies* s_2 by demonstrating that $(\neg s_2)$ *implies* $(\neg s_1)$ holds true. We also often use a *proof by contradiction* based on the law saying $((\neg s_2)$ *and* $s_1)$ *implies* **0** is true. Less formally, if from the assumption that s_2 is false and s_1 is true, then we obtain a false statement, s_1 *implies* s_2 is true. A *proof by induction* demonstrates that a statement s_i is true for all integers $i \geq b$, where b is a nonnegative integer. In general, a proof of this kind is made in the following way:

Basis. Prove that s_b is true.

Inductive Hypothesis. Suppose that there exists an integer n such that $n \geq b$ and s_m is true for all $b \leq m \leq n$.

Inductive Step. Prove that s_{n+1} is true under the assumption that the inductive hypothesis holds.

A proof by contradiction and a proof by induction are illustrated in the beginning of Section 1.2 (see Example 1.2).

1.2 Sets and Sequences

A *set*, Σ, is a collection of elements that are taken from some prespecified *universe*. If Σ contains an element a, then we symbolically write $a \in \Sigma$ and refer to a as a *member* of Σ. On the other hand, if a is not in Σ, then we write $a \notin \Sigma$. If Σ has a finite number of members, then

Σ is a *finite set*; otherwise, Σ is an *infinite set*. The finite set that has no member is the *empty set*, denoted by $\varnothing$. The *cardinality* of a finite set, Σ, *card*(Σ), is the number of Σ's members; note that *card*($\varnothing$) = 0.

Convention 1.2 Throughout this book, $\mathbb{N}$ denotes the set of *natural numbers*—that is, $\mathbb{N} = \{1, 2, \ldots\}$, and ${}_0\mathbb{N} = \{0\} \cup \mathbb{N}$. ■

Example 1.2 The purpose of this example is twofold. First, we give examples of sets. Second, as pointed out in the conclusion of Section 1.1, we illustrate how to make proofs by contradiction and by induction.

Let P be the set of all primes (a natural number n is prime if its only positive divisors are 1 and n).

A proof by contradiction. By contradiction, we next prove that P is infinite. That is, assume that P is finite. Set $k = card(P)$. Thus, P contains k numbers, $p_1, p_2, \ldots, p_k$. Set $n = p_1 p_2 \ldots p_k + 1$. Observe that n is not divisible by any p_i, $1 \leq i \leq k$. As a result, either n is a new prime or n equals a product of new primes. In either case, there exists a prime out of P, which contradicts that P contains all primes. Thus, P is infinite. Another proof by contradiction is given in Example 1.3.

A proof by induction. As already stated, *by induction*, we prove that a statement s_i holds for all $i \geq b$, where $b \in \mathbb{N}$. To illustrate, consider $\{i^2 | \ i \in \mathbb{N}\}$, and let s_i state

$$1 + 3 + 5 + \ldots + 2i - 1 = i^2$$

for all $i \in \mathbb{N}$; in other words, s_i says that the sum of odd integers is a perfect square. An inductive proof of this statement is as follows:

Basis. As $1 = 1^2$, s_1 is true.

Inductive Hypothesis. Assume that s_m is true for all $1 \leq m \leq n$, where n is a natural number.

Inductive Step. Consider $s_{n+1} = 1 + 3 + 5 + \ldots + (2n - 1) + (2(n + 1) - 1) = (n + 1)^2$. By the inductive hypothesis, $s_n = 1 + 3 + 5 + \ldots + (2n - 1) = n^2$. Hence, $1 + 3 + 5 + \ldots + (2n - 1) + (2(n + 1) - 1) = n^2 + 2n + 1 = (n + 1)^2$. Consequently, s_{n+1} holds, and the inductive proof is completed.

A finite set, Σ, is customarily specified by listing its members; that is, $\Sigma = \{a_1, a_2, \ldots, a_n\}$, where a_1 through a_n are all members of Σ; as a special case, we have $\{\} = \varnothing$. An infinite set, Ω, is usually specified by a property, π, so that Ω contains all elements satisfying π; in symbols, this specification has the following general format $\Omega = \{a | \ \pi(a)\}$. Sets whose members are other sets are usually called *families* of sets rather than sets of sets.

Let Σ and Ω be two sets. Σ is a *subset* of Ω, symbolically written as $\Sigma \subseteq \Omega$, if each member of Σ also belongs to Ω. Σ is a *proper subset* of Ω, written as $\Sigma \subset \Omega$, if $\Sigma \subseteq \Omega$ and Ω contains an element that is not in Σ. If $\Sigma \subseteq \Omega$ and $\Omega \subseteq \Sigma$, then Σ *equals* Ω, denoted by $\Sigma = \Omega$. The *power set* of Σ, denoted by *power*(Σ), is the set of all subsets of Σ.

For two sets, Σ and Ω, their *union*, *intersection*, and *difference* are denoted by $\Sigma \cup \Omega$, $\Sigma \cap \Omega$, and $\Sigma - \Omega$, respectively, and defined as $\Sigma \cup \Omega = \{a | \ a \in \Sigma \text{ or } a \in \Omega\}$, $\Sigma \cap \Omega = \{a | \ a \in \Sigma \text{ and } a \in \Omega\}$, and $\Sigma - \Omega = \{a | \ a \in \Sigma \text{ and } a \notin \Omega\}$. If Σ is a set over a universe U, then the *complement* of Σ is denoted by $\sim\Sigma$ and defined as $\sim\Sigma = U - \Sigma$. The operations of union, intersection, and complement are related by *DeMorgan's laws* stating that $\sim(\sim\Sigma \cup \sim\Omega) = \Sigma \cap \Omega$ and $\sim(\sim\Sigma \cap \sim\Omega) = \Sigma \cup \Omega$, for any two sets Σ and Ω. If $\Sigma \cap \Omega = \varnothing$, then Σ and Ω are *disjoint*. More generally, n sets $\Delta_1, \Delta_2, \ldots, \Delta_n$, where $n \geq 2$, are *pairwise disjoint* if $\Delta_i \cap \Delta_j = \varnothing$ for all $1 \leq i, j \leq n$ such that $i \neq j$.

A *sequence* is a list of elements from some universe. A sequence is *finite* if it consists of finitely many elements; otherwise, it is *infinite*. The *length of* a finite sequence x, denoted by $|x|$, is the number of elements in x. The *empty sequence*, denoted by ε, is the sequence consisting of no element; that is, $|\varepsilon| = 0$. For brevity, finite sequences are specified by listing their elements throughout. For instance, (0, 1, 0, 0) is shortened to 0100; notice that $|0100| = 4$.

1.3 Relations

For two objects, a and b, (a, b) denotes the *ordered pair* consisting of a and b in this order. Let A and B be two sets. The *Cartesian product* of A and B, $A \times B$, is defined as $A \times B = \{(a, b) | \ a \in A$ and $b \in B\}$. A *binary relation* or, briefly, a *relation*, ρ, from A to B is any subset of $A \times B$; that is, $\rho \subseteq A \times B$. If ρ represents a finite set, then it is a *finite relation*; otherwise, it is an *infinite relation*. The *domain of* ρ, denoted by *domain*(ρ), and the *range of* ρ, denoted by *range*(ρ), are defined as *domain*(ρ) $= \{a | \ (a, b) \in \rho$ for some $b \in B\}$ and *range*(ρ) $= \{b | \ (a, b) \in \rho$ for some $a \in A\}$. If $A = B$, then ρ is a *relation on* A. A relation σ is a *subrelation* of ρ if $\sigma \subseteq \rho$. The *inverse of* ρ, denoted by *inverse*(ρ), is defined as *inverse*(ρ) $= \{(b, a) | \ (a, b) \in \rho\}$. A *function* from A to B is a relation φ from A to B such that for every $a \in A$, *card*($\{b | \ b \in B$ and $(a, b) \in \varphi\}$) ≤ 1. If *domain*(φ) $= A$, then φ is *total*. If we want to emphasize that φ may not satisfy *domain*(φ) $= A$, then we say that φ is *partial*. If for every $b \in B$, *card*($\{a | \ a \in A$ and $(a, b) \in \varphi\}$) ≤ 1, then φ is an *injection*. If for every $b \in B$, *card*($\{a | \ a \in A$ and $(a, b) \in \varphi\}$) ≥ 1, then φ is a *surjection*. If φ is a total function that is both a surjection and an injection, then φ represents a *bijection*.

As relations and functions are defined as sets, the set operations allied to them, too. For instance, if $\rho \subseteq A \times B$ is a function, then its complement, $\sim\rho$, is defined as $(A \times B) - \rho$.

Convention 1.3 Let $\rho \subseteq A \times B$ be a relation. To express that $(a, b) \in \rho$, we usually write $a\rho b$. If ρ represents a function, then we often write $\rho(a) = b$ instead of $a\rho b$. If $\rho(a) = b$, then b is the *value* of ρ for *argument* a. ■

If there is a bijection from an infinite set Ψ to an infinite set Ξ, then Ψ and Ξ have the *same cardinality*. An infinite set, Ω, is *countable* or, synonymously, *enumerable*, if Ω and $\mathbb{N}$ have the same cardinality; otherwise, it is *uncountable* (according to Convention 1.2, $\mathbb{N}$ is the set of natural numbers).

Example 1.3 Consider the set of all even natural numbers, E. Define the bijection $\varphi(i) = 2i$, for all $i \in \mathbb{N}$. Observe that φ represents a bijection from $\mathbb{N}$ to E, so they have the same cardinality. Thus, E is countable.

Consider the set ς of all functions mapping $\mathbb{N}$ to $\{\mathbf{0}, \mathbf{1}\}$. By contradiction, we prove that ς is uncountable. Suppose that ς is countable. Thus, there is a bijection from ς to $\mathbb{N}$. Let ${}_if$ be the function mapped to the ith positive integer, for all $i \geq 1$. Consider the total function g from $\mathbb{N}$ to $\{\mathbf{0}, \mathbf{1}\}$ defined as $g(j) = \mathbf{0}$ if and only if ${}_jf(j) = \mathbf{1}$, for all $i \geq 1$, so $g(j) = \mathbf{1}$ if and only if ${}_jf(j) = \mathbf{0}$. As ς contains g, $g = {}_kf$ for some $k \geq 1$. Specifically, $g(k) = {}_kf(k)$. However, $g(k) = \mathbf{0}$ if and only if ${}_kf(k) = \mathbf{1}$, so $g(k) \neq {}_kf(k)$, which contradicts $g(k) = {}_kf(k)$. Thus, ς is uncountable.

The proof technique by which we have demonstrated that ς is uncountable is customarily called *diagonalization*. To see why, imagine an infinite table with ${}_1f$, ${}_2f$, … listed down the rows and 1, 2, … listed across the columns (see Figure 1.2). Each entry contains either **0** or **1**. Specifically, the

	1	2	…	k	…
${}_1f$	**0**	**1**		**0**	
${}_2f$	**1**	**1**		**1**	
⋮					
$g = {}_kf$	**0**	**0**		**0** *iff* **1**	
⋮					

Figure 1.2 Diagonalization.

entry in row ${}_if$ and column j contains **1** if and only if ${}_if(j) = \mathbf{1}$, so this entry contains **0** if and only if ${}_if(j) = \mathbf{0}$. A contradiction occurs at the diagonal entry in row ${}_kf$ and column k because $g(k) = \mathbf{0}$ if and only if ${}_kf(k) = \mathbf{1}$ and $g(k) = {}_kf(k)$; in other words, this diagonal entry contains **0** if and only if it contains **1**, which is impossible. We make use of this proof technique several times in this book.

Let A be a set, ρ be a relation on A, and $a, b \in A$. For $k \geq 1$, the *k-fold product* of ρ, ρ^k, is recursively defined as (1) $a\rho^1 b$ iff $a\rho b$, and (2) $a\rho^k b$ iff there exists $c \in A$ such that $a\rho c$ and $c\rho^{k-1} b$, for $k \geq 2$. Furthermore, $a\rho^0 b$ if and only if $a = b$. The *transitive closure* of ρ, ρ^+, is defined as $a\rho^+ b$ if and only if $a\rho^k b$, for some $k \geq 1$, and the *reflexive and transitive closure* of ρ, ρ^*, is defined as $a\rho^* b$ if and only if $a\rho^k b$, for some $k \geq 0$.

1.4 Graphs

Let A be a set. A *directed* graph or, briefly, a *graph* is a pair $G = (A, \rho)$, where ρ is a relation on A. Members of A are called *nodes*, and ordered pairs in ρ are called *edges*. If $(a, b) \in \rho$, then edge (a, b) *leaves* a and *enters* b. Let $a \in A$; then, the *in-degree* of a and the *out-degree* of a are $card(\{b | (b, a) \in \rho\})$ and $card(\{c | (a, c) \in \rho\})$. A sequence of nodes, $(a_0, a_1, \ldots, a_n)$, where $n \geq 1$, is a *path of length n* from a_0 to a_n if $(a_{i-1}, a_i) \in \rho$ for all $1 \leq i \leq n$; if, in addition, $a_0 = a_n$, then $(a_0, a_1, \ldots, a_n)$ is a *cycle of length n*. In this book, we frequently *label* the edges of G with some attached information. Pictorially, we represent $G = (A, \rho)$ so we draw each edge $(a, b) \in \rho$ as an arrow from a to b possibly with its label as illustrated in Example 1.4.

Example 1.4 Consider a program p and its *call graph* $G = (P, \rho)$, where P represents the set of subprograms in p, and $(x, y) \in \rho$ iff subprogram x calls subprogram y. Specifically, let $P = \{a, b, c, d\}$, and $\rho = \{(a, b), (a, c), (b, d), (c, d)\}$, which says a calls b and c, b calls d, and c calls d as well (see Figure 1.3).

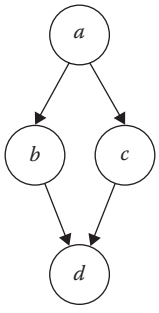

Figure 1.3 Graph.

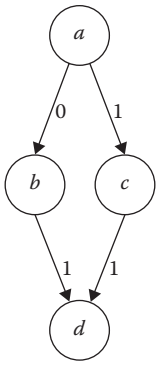

Figure 1.4 Labeled graph.

The in-degree of a is 0, and its out-degree is 2. Notice that (a, b, d) is a path of length 2 in G. G contains no cycle because none of its paths starts and ends in the same node.

Suppose we use G to study the value of a global variable during the four calls. Specifically, we want to express that this value is zero when call (a, b) occurs; otherwise, it is one. We express this by labeling the edges of G in the way given in Figure 1.4.

Let $G = (A, \rho)$ be a graph. G is an *acyclic graph* if it contains no cycle. If $(a_0, a_1, \ldots, a_n)$ is a path in G, then a_0 is an *ancestor* of a_n and a_n is a *descendant* of a_0; if in addition, $n = 1$, then a_0 is a *direct ancestor* of a_n and a_n a *direct descendant* of a_0. A *tree* is an acyclic graph $T = (A, \rho)$ such that A contains a specified node, called the *root* of T and denoted by $root(T)$, and every $a \in A - root(T)$ is a descendant of $root(T)$ and its in-degree is one. If $a \in A$ is a node whose out-degree is 0, then a is a *leaf*; otherwise, it is an *interior node*. In this book, a tree T is always considered as an *ordered tree* in which each interior node $a \in A$ has all its direct descendants, b_1 through b_n, where $n \geq 1$, ordered from the left to the right so that b_1 is the leftmost direct descendant of a and b_n is the rightmost direct descendant of a. At this point, a is the *parent* of its *children* b_1 through b_n, and all these nodes together with the edges connecting them, (a, b_1) through (a, b_n), are called a *parent-children portion of T*. The *frontier* of T, denoted by $frontier(T)$, is the sequence of T's leaves ordered from the left to the right. The *depth* of T, $depth(T)$, is the length of the longest path in T. A tree $S = (B, \upsilon)$ is a *subtree of* T if $\emptyset \subset B \subseteq A$, $\upsilon \subseteq \rho \cap (B \times B)$, and in T, no node in $A - B$ is a descendant of a node in B; S is an *elementary subtree* of T if $depth(S) = 1$.

Like any graph, a tree T can be described as a two-dimensional structure. To simplify this description, however, we draw a tree T with its root on the top and with all edges directed down. Each parent has its children drawn from the left to the right according to its ordering. Drawing T in this way, we always omit all arrowheads.

Apart from this two-dimensional representation, however, it is frequently convenient to specify T by a one-dimensional representation, denoted by $odr(T)$, in which each subtree of T is represented by the expression appearing inside a balanced pair of $\langle$ and $\rangle$ with the node that is the root of that subtree appearing immediately to the left of $\langle$. More precisely, $odr(T)$ is defined by the following recursive rules:

1. If T consists of a single node a, then $odr(T) = a$.
2. Let (a, b_1) through (a, b_n), where $n \geq 1$, be the parent-children portion of T, $root(T) = a$, and T_k be the subtree rooted at b_k, $1 \leq k \leq n$, then $odr(T) = a\langle odr(T_1)\ odr(T_2)\ \ldots\ odr(T_n)\rangle$.

Example 1.5 illustrates both the one-dimensional *odr*-representation and the two-dimensional pictorial representation of a tree. For brevity, we prefer the former throughout this book.

Example 1.5 Graph G discussed in Example 1.4 is acyclic. However, it is no tree because the in-degree of node d is two. By removing edge (b, d), we obtain a tree $T = (P, \tau)$, where $P = \{a, b, c, d\}$ and $\tau = \{(a, b), (a, c), (c, d)\}$. Nodes a and c are interior nodes while b and d are leaves. The root of T is a. We define b and c as the first and the second child of a, respectively. A parent-children portion of T is, for instance, (a, b) and (a, c). Notice that $frontier(T) = bd$, and $depth(T) = 2$. Following the recursive rules (1) and (2), we obtain the one-dimensional representation of T as $odr(T) = a\langle bc\langle d\rangle\rangle$. Its subtrees are $a\langle bc\langle d\rangle\rangle$, $c\langle d\rangle$, b, and d. In Figure 1.5, we pictorially describe $a\langle bc\langle d\rangle\rangle$ and $c\langle d\rangle$.

Exercises

1. A *tautology* is a statement that is true for all possible truth values of the statement variables.
   a. Prove that the contrapositive law represents a tautology.
   b. State and prove five more tautologies.
   c. Finally, from a more general viewpoint, prove that every theorem of a formal mathematical system represents a tautology, and conversely, every tautology is a theorem.
2. A *Boolean algebra* is a formal mathematical system, which consists of a set Σ and operations $\vee$, $\wedge$, and $\neg$. The axioms of Boolean algebra are as follows.
   Associativity:

$$a \vee (b \vee c) = (a \vee b) \vee c, \text{ and } a \wedge (b \wedge c) = (a \wedge b) \wedge c, \text{ for all } a, b, c \in \Sigma.$$

   Commutativity:

$$a \vee b = b \vee a, \text{ and } a \wedge b = b \wedge a, \text{ for all } a, b \in \Sigma.$$

   Distributivity:

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c), \text{ and } a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c), \text{ for all } a, b \in \Sigma.$$

   In addition, Σ contains two distinguished members, 0 and 1, such that for all $a \in \Sigma$,

$$a \vee 0 = a,\ a \wedge 1 = a,\ a \vee (\neg a) = 1,\ a \wedge (\neg a) = 0$$

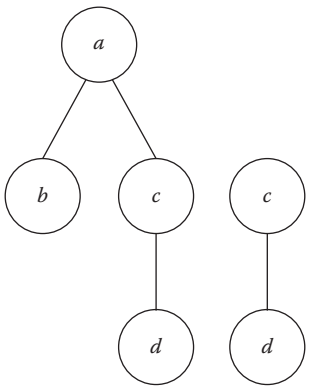

Figure 1.5 A tree and a subtree.

The rule of inference is substitution of equals for equals.

a. Consider the Boolean algebra in which $0 = \mathbf{0}$ and $1 = \mathbf{1}$, where **1** and **0** denote truth and falsity according to Convention 1.1, and $\Sigma - \{\mathbf{1}, \mathbf{0}\} = \varnothing$. Furthermore, consider this statement

$$a \vee (b \wedge \neg a) = a \vee b$$

where $a, b \in \Sigma$. Prove that this statement represents a theorem in the Boolean algebra.

b. Reformulate (a) so Σ is any superset of $\{\mathbf{1}, \mathbf{0}\}$. Does the above-mentioned statement necessarily represent a theorem in the Boolean algebra generalized in this way? Prove your answer rigorously.

c. Give five statements and prove that they are theorems in terms of the Boolean algebra generalized in (b).

3. By induction, prove that for any set Σ, $card(power(\Sigma)) = 2^{card(\Sigma)}$ (see Section 1.1 for $power(\Sigma)$).

4. Let $\Sigma \subseteq {}_0\mathbb{N}$, and let φ be the total function from ${}_0\mathbb{N}$ to $\{\mathbf{0}, \mathbf{1}\}$ defined by $\varphi(i) = \mathbf{1}$ iff $i \in \Sigma$, for all $i \in {}_0\mathbb{N}$; then, φ is the *characteristic function* of Σ. Express basic set operations, such as union, in terms of characteristic functions.

5. Let Σ and Ω be two sets, and let ρ and ρ' be two relations from Σ to Ω. If ρ and ρ' represent two identical subsets of $\Sigma \times \Omega$, then ρ *equals* ρ', symbolically written as $\rho = \rho'$.
Perform (a) and (b), given as follows.
   a. Illustrate this definition by five examples in terms of relations over ${}_0\mathbb{N}$.
   b. Reformulate this definition by using characteristic functions.

6. Let Σ be a set, and let ρ be a relation on Σ. Then,
   a. If for all $a \in \Sigma$, $a\rho a$, then ρ is *reflexive.*
   b. If for all $a, b \in \Sigma$, $a\rho b$ implies $b\rho a$, then ρ is *symmetric.*
   c. If for all $a, b \in \Sigma$, ($a\rho b$ and $b\rho a$) implies $a = b$, then ρ is *antisymmetric.*
   d. If for all $a, b, c \in \Sigma$, ($a\rho b$ and $b\rho c$) implies $a\rho c$, then ρ is *transitive.*

   Consider relations (i) through (ix), given as follows. For each of them, determine whether it is reflexive, symmetric, antisymmetric, or transitive.
   i. $\varnothing$
   ii. $\{(1, 3), (3, 1), (8, 8)\}$
   iii. $\{(1, 1), (2, 2), (8, 8)\}$
   iv. $\{(x, x) | \ x \in \Sigma\}$
   v. $\{(x, y) | \ x, y \in \Sigma, x < y\}$
   vi. $\{(x, y) | \ x, y \in \Sigma, x \leq y\}$
   vii. $\{(x, y) | \ x, y \in \Sigma, x + y = 9\}$
   viii. $\{(x, y) | \ x, y \in \Sigma, y$ is divisible by $x\}$
   ix. $\{(x, y) | \ x, y \in \Sigma, x - y$ is divisible by $3\}$

   Note that x is divisible by y if there exists a positive integer z such that $x = yz$.

7. Let Σ be a set, and let ρ be a relation on Σ. If ρ is reflexive, symmetric, and transitive, then ρ is an *equivalence relation*. Let ρ be an equivalence relation on Σ. Then, ρ partitions Σ into disjoint subsets, called *equivalence classes*, so that for each $a \in \Sigma$, the equivalence class of a is denoted by $[a]$ and defined as $[a] = \{b | \ a\rho b\}$.

   Prove that for all a and b in Σ, either $[a] = [b]$ or $[a] \cap [b] = \varnothing$.

8. Let Σ be a set, and let ρ be a relation on Σ. If ρ is reflexive, antisymmetric, and transitive, then ρ is a *partial order*. If ρ is a partial order satisfying either $a\rho b$ or $b\rho a$, for all $a, b \in \Sigma$ such that $a \neq b$, then ρ is a *linear order*.

   Let Σ be a set. Define the relation ρ on $power(\Sigma)$ as $\rho = \{(A, B) | \ A, B \in power(\Sigma), A \subseteq B\}$ (see Section 1.1 for $power(\Sigma)$). Prove that ρ represents a partial order.

9 S. Prove the following two theorems.

Theorem Let Σ be a set, ρ be a relation on Σ, and ρ^+ be the transitive closure of ρ. Then, (i) ρ^+ is a transitive relation, and (ii) if ρ' is a transitive relation such that $\rho \subseteq \rho'$, then $\rho^+ \subseteq \rho'$. ■