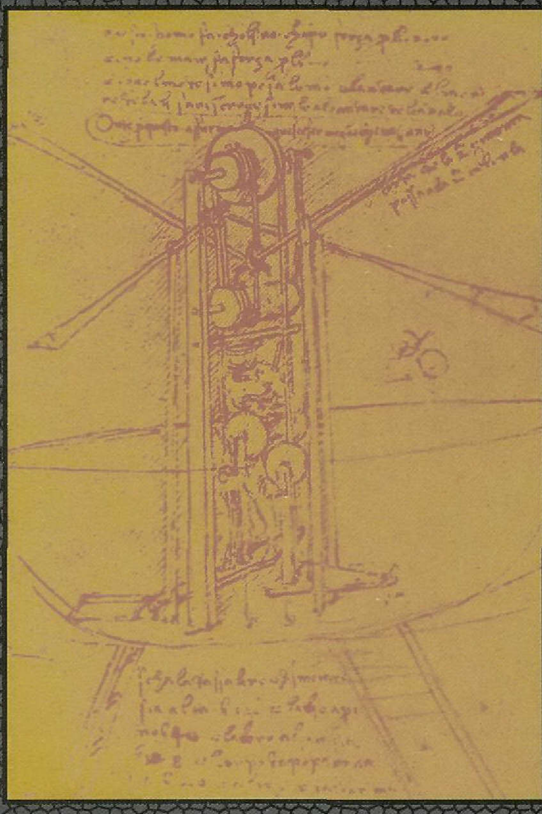
THOMSON
COURSE TECHNOLOGY

Introduction to the Theory of COMPUTATION

Second Edition



MICHAEL SIPSER

INTRODUCTION TO THE THEORY OF COMPUTATION, SECOND EDITION

MICHAEL SIPSER

Massachusetts Institute of Technology

THOMSON

COURSE TECHNOLOGY

Australia • Canada • Mexico • Singapore • Spain • United Kingdom • United States

THOMSON

COURSE TECHNOLOGY

Introduction to the Theory of Computation,
Second Edition

by Michael Sipser

Senior Product Manager:
Alyssa Pratt

Executive Editor:
Mac Mendelsohn

Associate Production Manager:
Aimee Poirier

Senior Marketing Manager:
Karen Seitz

Associate Product Manager:
Mirella Misiaszek

Editorial Assistant:
Jennifer Smith

Senior Manufacturing Coordinator:
Trevor Kallop

Cover Designer:
Steve Deschesne

Printed in the United States of America

1 2 3 4 5 6 7 8 9 QWT 09 08 07 06 05

For more information, contact
Thomson Course Technology
25 Thomson Place
Boston, Massachusetts, 02210.

Or find us on the World Wide Web at:
www.course.com

ISBN 0-534-95097-3

CONTENTS

PREFACE TO THE FIRST EDITION

TO THE STUDENT

Welcome!

You are about to embark on the study of a fascinating and important subject: the theory of computation. It comprises the fundamental mathematical properties of computer hardware, software, and certain applications thereof. In studying this subject we seek to determine what can and cannot be computed, how quickly, with how much memory, and on which type of computational model. The subject has obvious connections with engineering practice, and, as in many sciences, it also has purely philosophical aspects.

I know that many of you are looking forward to studying this material but some may not be here out of choice. You may want to obtain a degree in computer science or engineering, and a course in theory is required—God knows why. After all, isn't theory arcane, boring, and worst of all, irrelevant?

To see that theory is neither arcane nor boring, but instead quite understandable and even interesting, read on. Theoretical computer science does have many fascinating big ideas, but it also has many small and sometimes dull details that can be tiresome. Learning any new subject is hard work, but it becomes easier and more enjoyable if the subject is properly presented. My primary objective in writing this book is to expose you to the genuinely exciting aspects of computer theory, without getting bogged down in the drudgery. Of course, the only way to determine whether theory interests you is to try learning it.

Theory is relevant to practice. It provides conceptual tools that practitioners use in computer engineering. Designing a new programming language for a specialized application? What you learned about *grammars* in this course comes in handy. Dealing with string searching and pattern matching? Remember *finite automata* and *regular expressions*. Confronted with a problem that seems to require more computer time than you can afford? Think back to what you learned about *NP-completeness*. Various application areas, such as modern cryptographic protocols, rely on theoretical principles that you will learn here.

Theory also is relevant to you because it shows you a new, simpler, and more elegant side of computers, which we normally consider to be complicated machines. The best computer designs and applications are conceived with elegance in mind. A theoretical course can heighten your aesthetic sense and help you build more beautiful systems.

Finally, theory is good for you because studying it expands your mind. Computer technology changes quickly. Specific technical knowledge, though useful today, becomes outdated in just a few years. Consider instead the abilities to think, to express yourself clearly and precisely, to solve problems, and to know when you haven't solved a problem. These abilities have lasting value. Studying theory trains you in these areas.

Practical considerations aside, nearly everyone working with computers is curious about these amazing creations, their capabilities, and their limitations. A whole new branch of mathematics has grown up in the past 30 years to answer certain basic questions. Here's a big one that remains unsolved: If I give you a large number, say, with 500 digits, can you find its factors (the numbers that divide it evenly), in a reasonable amount of time? Even using a supercomputer, no one presently knows how to do that in all cases *within the lifetime of the universe!* The factoring problem is connected to certain secret codes in modern cryptosystems. Find a fast way to factor and fame is yours!

TO THE EDUCATOR

This book is intended as an upper-level undergraduate or introductory graduate text in computer science theory. It contains a mathematical treatment of the subject, designed around theorems and proofs. I have made some effort to accommodate students with little prior experience in proving theorems, though more experienced students will have an easier time.

My primary goal in presenting the material has been to make it clear and interesting. In so doing, I have emphasized intuition and "the big picture" in the subject over some lower level details.

For example, even though I present the method of proof by induction in Chapter 0 along with other mathematical preliminaries, it doesn't play an important role subsequently. Generally I do not present the usual induction proofs of the correctness of various constructions concerning automata. If presented clearly, these constructions convince and do not need further argument. An induction may confuse rather than enlighten because induction itself is a rather sophisticated technique that many find mysterious. Belaboring the obvious with an induction risks teaching students that mathematical proof is a formal manipulation instead of teaching them what is and what is not a cogent argument.

A second example occurs in Parts Two and Three, where I describe algorithms in prose instead of pseudocode. I don't spend much time programming Turing machines (or any other formal model). Students today come with a programming background and find the Church–Turing thesis to be self-evident. Hence I don't present lengthy simulations of one model by another to establish their equivalence.

Besides giving extra intuition and suppressing some details, I give what might be called a classical presentation of the subject material. Most theorists will find the choice of material, terminology, and order of presentation consistent with that of other widely used textbooks. I have introduced original terminology in only a few places, when I found the standard terminology particularly obscure or confusing. For example I introduce the term *mapping reducibility* instead of *many–one reducibility*.

Practice through solving problems is essential to learning any mathematical subject. In this book, the problems are organized into two main categories called *Exercises* and *Problems*. The Exercises review definitions and concepts. The Problems require some ingenuity. Problems marked with a star are more difficult. I have tried to make both the Exercises and Problems interesting challenges.

THE FIRST EDITION

Introduction to the Theory of Computation first appeared as a Preliminary Edition in paperback. The first edition differs from the Preliminary Edition in several substantial ways. The final three chapters are new: Chapter 8 on space complexity; Chapter 9 on provable intractability; and Chapter 10 on advanced topics in complexity theory. Chapter 6 was expanded to include several advanced topics in computability theory. Other chapters were improved through the inclusion of additional examples and exercises.

Comments from instructors and students who used the Preliminary Edition were helpful in polishing Chapters 0–7. Of course, the errors they reported have been corrected in this edition.

Chapters 6 and 10 give a survey of several more advanced topics in computability and complexity theories. They are not intended to comprise a cohesive unit in the way that the remaining chapters are. These chapters are included to allow the instructor to select optional topics that may be of interest to the serious student. The topics themselves range widely. Some, such as *Turing reducibility* and *alternation*, are direct extensions of other concepts in the book. Others, such as *decidable logical theories* and *cryptography*, are brief introductions to large fields.

FEEDBACK TO THE AUTHOR

The internet provides new opportunities for interaction between authors and readers. I have received much e-mail offering suggestions, praise, and criticism, and reporting errors for the Preliminary Edition. Please continue to correspond! I try to respond to each message personally, as time permits. The e-mail address for correspondence related to this book is

sipserbook@math.mit.edu.

A web site that contains a list of errata is maintained. Other material may be added to that site to assist instructors and students. Let me know what you would like to see there. The location for that site is

http://www-math.mit.edu/~sipser/book.html.

ACKNOWLEDGMENTS

I could not have written this book without the help of many friends, colleagues, and my family.

I wish to thank the teachers who helped shape my scientific viewpoint and educational style. Five of them stand out. My thesis advisor, Manuel Blum, is due a special note for his unique way of inspiring students through clarity of thought, enthusiasm, and caring. He is a model for me and for many others. I am grateful to Richard Karp for introducing me to complexity theory, to John Addison for teaching me logic and assigning those wonderful homework sets, to Juris Hartmanis for introducing me to the theory of computation, and to my father for introducing me to mathematics, computers, and the art of teaching.

This book grew out of notes from a course that I have taught at MIT for the past 15 years. Students in my classes took these notes from my lectures. I hope they will forgive me for not listing them all. My teaching assistants over the years, Avrim Blum, Thang Bui, Andrew Chou, Benny Chor, Stavros Cosmadakis, Aditi Dhagat, Wayne Goddard, Parry Husbands, Dina Kravets, Jakov Kučan, Brian O'Neill, Ioana Popescu, and Alex Russell, helped me to edit and expand these notes and provided some of the homework problems.

Nearly three years ago, Tom Leighton persuaded me to write a textbook on the theory of computation. I had been thinking of doing so for some time, but it took Tom's persuasion to turn theory into practice. I appreciate his generous advice on book writing and on many other things.

I wish to thank Eric Bach, Peter Beebee, Cris Calude, Marek Chrobak, Anna Chefter, Guang-Ien Cheng, Elias Dahlhaus, Michael Fischer, Steve Fisk, Lance Fortnow, Henry J. Friedman, Jack Fu, Seymour Ginsburg, Oded Goldreich, Brian Grossman, David Harel, Micha Hofri, Dung T. Huynh, Neil Jones, H. Chad Lane, Kevin Lin, Michael Loui, Silvio Micali, Tadao Murata, Christos Papadimitriou, Vaughan Pratt, Daniel Rosenband, Brian Scassellati, Ashish Sharma, Nir Shavit, Alexander Shen, Ilya Shlyakhter, Matt Stallmann, Perry Susskind, Y. C. Tay, Joseph Traub, Osamu Watanabe, Peter Widmayer, David Williamson, Derick Wood, and Charles Yang for comments, suggestions, and assistance as the writing progressed.

The following people provided additional comments that have improved this book: Isam M. Abdelhameed, Eric Allender, Shay Artzi, Michelle Atherton, Rolfe Blodgett, Al Briggs, Brian E. Brooks, Jonathan Buss, Jin Yi Cai, Steve Chapel, David Chow, Michael Ehrlich, Yaakov Eisenberg, Farzan Fallah, Shaun Flisakowski, Hjalmtyr Hafsteinsson, C. R. Hale, Maurice Herlihy, Vegard Holmedahl, Sandy Irani, Kevin Jiang, Rhys Price Jones, James M. Jowdy, David M. Martin Jr., Manrique Mata-Montero, Ryota Matsuura, Thomas Minka, Farooq Mohammed, Tadao Murata, Jason Murray, Hideo Nagahashi, Kazuo Ohta, Constantine Papageorgiou, Joseph Raj, Rick Regan, Rhonda A. Reumann, Michael Rintzler, Arnold L. Rosenberg, Larry Roske, Max Rozenoer, Walter L. Ruzzo, Sanatan Sahgal, Leonard Schulman, Steve Seiden, Joel Seiferas, Ambuj Singh, David J. Stucki, Jayram S. Thathachar, H. Venkateswaran, Tom Whaley, Christopher Van Wyk, Kyle Young, and Kyoung Hwan Yun.

Robert Sloan used an early version of the manuscript for this book in a class that he taught and provided me with invaluable commentary and ideas from his experience with it. Mark Herschberg, Kazuo Ohta, and Latanya Sweeney read over parts of the manuscript and suggested extensive improvements. Shafi Goldwasser helped me with material in Chapter 10.

I received expert technical support from William Baxter at Superscript, who wrote the LaTeX macro package implementing the interior design, and from Larry Nolan at the MIT mathematics department, who keeps everything running.

It has been a pleasure to work with the folks at PWS Publishing in creating the final product. I mention Michael Sugarman, David Dietz, Elise Kaiser, Monique Calello, Susan Garland and Tanja Brull because I have had the most contact with them, but I know that many others have been involved, too. Thanks to Jerry Moore for the copy editing, to Diane Levy for the cover design, and to Catherine Hawkes for the interior design.

I am grateful to the National Science Foundation for support provided under grant CCR-9503322.

My father, Kenneth Sipser, and sister, Laura Sipser, converted the book diagrams into electronic form. My other sister, Karen Fisch, saved us in various computer emergencies, and my mother, Justine Sipser, helped out with motherly advice. I thank them for contributing under difficult circumstances, including insane deadlines and recalcitrant software.

Finally, my love goes to my wife, Ina, and my daughter, Rachel. Thanks for putting up with all of this.

Cambridge, Massachusetts **Michael Sipser**
October, 1996

PREFACE TO THE SECOND EDITION

Judging from the email communications that I've received from so many of you, the biggest deficiency of the first edition is that it provides no sample solutions to any of the problems. So here they are. Every chapter now contains a new *Selected Solutions* section that gives answers to a representative cross-section of that chapter's exercises and problems. To make up for the loss of the solved problems as interesting homework challenges, I've also added a variety of new problems. Instructors may request an Instructor's Manual that contains additional solutions by contacting the sales representative for their region designated at `www.course.com`.

A number of readers would have liked more coverage of certain "standard" topics, particularly the Myhill–Nerode Theorem and Rice's Theorem. I've partially accommodated these readers by developing these topics in the solved problems. I did not include the Myhill–Nerode Theorem in the main body of the text because I believe that this course should provide only an introduction to finite automata and not a deep investigation. In my view, the role of finite automata here is for students to explore a simple formal model of computation as a prelude to more powerful models, and to provide convenient examples for subsequent topics. Of course, some people would prefer a more thorough treatment, while others feel that I ought to omit all reference to (or at least dependence on) finite automata. I did not include Rice's Theorem in the main body of the text because, though it can be a useful "tool" for proving undecidability, some students might use it mechanically without really understanding what is going on. Using reductions instead, for proving undecidability, gives more valuable preparation for the reductions that appear in complexity theory.

I am indebted to my teaching assistants, Ilya Baran, Sergi Elizalde, Rui Fan, Jonathan Feldman, Venkatesan Guruswami, Prahladh Harsha, Christos Kapoutsis, Julia Khodor, Adam Klivans, Kevin Matulef, Ioana Popescu, April Rasala, Sofya Raskhodnikova, and Iuliu Vasilescu who helped me to craft some of the new problems and solutions. Ching Law, Edmond Kayi Lee, and Zulfikar Ramzan also contributed to the solutions. I thank Victor Shoup for coming up with a simple way to repair the gap in the analysis of the probabilistic primality algorithm that appears in the first edition.

I appreciate the efforts of the people at Course Technology in pushing me and the other parts of this project along, especially Alyssa Pratt and Aimee Poirier. Many thanks to Gerald Eisman, Weizhen Mao, Rupak Majumdar, Chris Umans, and Christopher Wilson for their reviews. I'm indebted to Jerry Moore for his superb job copy editing and to Laura Segel of ByteGraphics (`lauras@bytegraphics.com`) for her beautifully precise rendition of the figures.

The volume of email I've received has been more than I expected. Hearing from so many of you from so many places has been absolutely delightful, and I've tried to respond to all eventually—my apologies for those I missed. I've listed here the people who made suggestions that specifically affected this edition, but I thank everyone for their correspondence.

Luca Aceto, Arash Afkanpour, Rostom Aghanian, Eric Allender, Karun Bakshi, Brad Ballinger, Ray Bartkus, Louis Barton, Arnold Beckmann, Mihir Bellare, Kevin Trent Bergeson, Matthew Berman, Rajesh Bhatt, Somenath Biswas, Lenore Blum, Mauro A. Bonatti, Paul Bondin, Nicholas Bone, Ian Bratt, Gene Browder, Doug Burke, Sam Buss, Vladimir Bychkovsky, Bruce Carneal, Soma Chaudhuri, Rong-Jaye Chen, Samir Chopra, Benny Chor, John Clausen, Allison Coates, Anne Condon, Jeffrey Considine, John J. Crashell, Claude Crepeau, Shaun Cutts, Susheel M. Daswani, Geoff Davis, Scott Dexter, Peter Drake, Jeff Edmonds, Yaakov Eisenberg, Kurtcebe Eroglu, Georg Essl, Alexander T. Fader, Farzan Fallah, Faith Fich, Joseph E. Fitzgerald, Perry Fizzano, David Ford, Jeannie Fromer, Kevin Fu, Atsushi Fujioka, Michel Galley, K. Ganesan, Simson Garfinkel, Travis Gebhardt, Peymann Gohari, Ganesh Gopalakrishnan, Steven Greenberg, Larry Griffith, Jerry Grossman, Rudolf de Haan, Michael Halper, Nick Harvey, Mack Hendricks, Laurie Hiyakumoto, Steve Hockema, Michael Hoehle, Shahadat Hossain, Dave Isecke, Ghaith Issa, Raj D. Iyer, Christian Jacobi, Thomas Janzen, Mike D. Jones, Max Kanovitch, Aaron Kaufman, Roger Khazan, Sarfraz Khurshid, Kevin Killourhy, Seungjoo Kim, Victor Kuncak, Kanata Kuroda, Suk Y. Lee, Edward D. Legenski, Li-Wei Lehman, Kong Lei, Zsolt Lengvarszky, Jeffrey Levetin, Baekjun Lim, Karen Livescu, Thomas Lasko, Stephen Louie, TzerHung Low, Wolfgang Maass, Arash Madani, Michael Manapat, Wojciech Marchewka, David M. Martin Jr., Anders Martinson, Lyle McGeoch, Alberto Medina, Kurt Mehlhorn, Nihar Mehta, Albert R. Meyer, Thomas Minka, Mariya Minkova, Daichi Mizuguchi, G. Allen Morris III, Damon Mosk-Aoyama, Xiaolong Mou, Paul Muir, Ger-

man Muller, Donald Nelson, Gabriel Nivasch, Mary Obelnicki, Kazuo Ohta, Thomas M. Oleson, Jr., Curtis Oliver, Owen Ozier, Rene Peralta, Alexander Perlis, Holger Petersen, Detlef Plump, Robert Prince, David Pritchard, Bina Reed, Nicholas Riley, Ronald Rivest, Robert Robinson, Christi Rockwell, Phil Rogaway, Max Rozenoer, John Rupf, Teodor Rus, Larry Ruzzo, Brian Sanders, Cem Say, Kim Schioett, Joel Seiferas, Joao Carlos Setubal, Geoff Lee Seyon, Mark Skandera, Bob Sloan, Geoff Smith, Marc L. Smith, Stephen Smith, Alex C. Snoeren, Guy St-Denis, Larry Stockmeyer, Radu Stoleru, David Stucki, Hisham M. Sueyllam, Kenneth Tam, Elizabeth Thompson, Michel Toulouse, Eric Tria, Chittaranjan Tripathy, Dan Trubow, Hiroki Ueda, Giora Unger, Kurt L. Van Etten, Jesir Vargas, Bienvenido Velez-Rivera, Kobus Vos, Alex Vrenios, Sven Waibel, Marc Waldman, Tom Whaley, Anthony Widjaja, Sean Williams, Joseph N. Wilson, Chris Van Wyk, Guangming Xing, Vee Voon Yee, Cheng Yongxi, Neal Young, Timothy Yuen, Kyle Yung, Jinghua Zhang, Lilla Zollei.

Most of all I thank my family—Ina, Rachel, and Aaron—for their patience, understanding, and love as I sat for endless hours here in front of my computer screen.

Cambridge, Massachusetts **_Michael Sipser_**
December, 2004

P_k, we know that

$$P_{k+1} = P_k M - Y.$$

Therefore, using the induction hypothesis to calculate P_k,

$$P_{k+1} = \left[PM^k - Y\left(\frac{M^k - 1}{M - 1}\right)\right] M - Y.$$

Multiplying through by M and rewriting Y yields

$$P_{k+1} = PM^{k+1} - Y\left(\frac{M^{k+1} - M}{M - 1}\right) - Y\left(\frac{M - 1}{M - 1}\right)$$
$$= PM^{k+1} - Y\left(\frac{M^{k+1} - 1}{M - 1}\right).$$

Thus the formula is correct for $t = k + 1$, which proves the theorem.

Problem 0.14 asks you to use the preceding formula to calculate actual mortgage payments.

EXERCISES

0.1 Examine the following formal descriptions of sets so that you understand which members they contain. Write a short informal English description of each set.

- **a.** $\{1, 3, 5, 7, \dots\}$
- **b.** $\{\dots, -4, -2, 0, 2, 4, \dots\}$
- **c.** $\{n \mid n = 2m \text{ for some } m \text{ in } \mathcal{N}\}$
- **d.** $\{n \mid n = 2m \text{ for some } m \text{ in } \mathcal{N}, \text{ and } n = 3k \text{ for some } k \text{ in } \mathcal{N}\}$
- **e.** $\{w \mid w \text{ is a string of 0s and 1s and } w \text{ equals the reverse of } w\}$
- **f.** $\{n \mid n \text{ is an integer and } n = n + 1\}$

0.2 Write formal descriptions of the following sets

- **a.** The set containing the numbers 1, 10, and 100
- **b.** The set containing all integers that are greater than 5
- **c.** The set containing all natural numbers that are less than 5
- **d.** The set containing the string aba
- **e.** The set containing the empty string
- **f.** The set containing nothing at all

0.3 Let A be the set $\{\mathtt{x}, \mathtt{y}, \mathtt{z}\}$ and B be the set $\{\mathtt{x}, \mathtt{y}\}$.

a. Is A a subset of B?

b. Is B a subset of A?

c. What is $A \cup B$?

d. What is $A \cap B$?

e. What is $A \times B$?

f. What is the power set of B?

0.4 If A has a elements and B has b elements, how many elements are in $A \times B$? Explain your answer.

0.5 If C is a set with c elements, how many elements are in the power set of C? Explain your answer.

0.6 Let X be the set $\{1, 2, 3, 4, 5\}$ and Y be the set $\{6, 7, 8, 9, 10\}$. The unary function $f\colon X \longrightarrow Y$ and the binary function $g\colon X \times Y \longrightarrow Y$ are described in the following tables.

n	$f(n)$
1	6
2	7
3	6
4	7
5	6

g	6	7	8	9	10
1	10	10	10	10	10
2	7	8	9	10	6
3	7	7	8	8	9
4	9	8	7	6	10
5	6	6	6	6	6

a. What is the value of $f(2)$?

b. What are the range and domain of f?

c. What is the value of $g(2, 10)$?

d. What are the range and domain of g?

e. What is the value of $g(4, f(4))$?

0.7 For each part, give a relation that satisfies the condition.

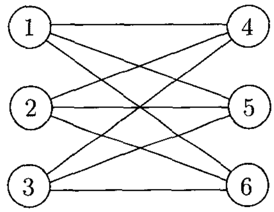
a. Reflexive and symmetric but not transitive

b. Reflexive and transitive but not symmetric

c. Symmetric and transitive but not reflexive

0.8 Consider the undirected graph $G = (V, E)$ where V, the set of nodes, is $\{1, 2, 3, 4\}$ and E, the set of edges, is $\{\{1, 2\}, \{2, 3\}, \{1, 3\}, \{2, 4\}, \{1, 4\}\}$. Draw the graph G. What is the degree of node 1? of node 3? Indicate a path from node 3 to node 4 on your drawing of G.

0.9 Write a formal description of the following graph.



PROBLEMS

0.10 Find the error in the following proof that $2 = 1$.
Consider the equation $a = b$. Multiply both sides by a to obtain $a^2 = ab$. Subtract b^2 from both sides to get $a^2 - b^2 = ab - b^2$. Now factor each side, $(a+b)(a-b) = b(a-b)$, and divide each side by $(a-b)$, to get $a + b = b$. Finally, let a and b equal 1, which shows that $2 = 1$.

0.11 Find the error in the following proof that all horses are the same color.
CLAIM: In any set of h horses, all horses are the same color.
PROOF: By induction on h.

Basis: For $h = 1$. In any set containing just one horse, all horses clearly are the same color.

Induction step: For $k \geq 1$ assume that the claim is true for $h = k$ and prove that it is true for $h = k + 1$. Take any set H of $k + 1$ horses. We show that all the horses in this set are the same color. Remove one horse from this set to obtain the set H_1 with just k horses. By the induction hypothesis, all the horses in H_1 are the same color. Now replace the removed horse and remove a different one to obtain the set H_2. By the same argument, all the horses in H_2 are the same color. Therefore all the horses in H must be the same color, and the proof is complete.

0.12 Show that every graph with 2 or more nodes contains two nodes that have equal degrees.

A***0.13** **Ramsey's theorem.** Let G be a graph. A ***clique*** in G is a subgraph in which every two nodes are connected by an edge. An ***anti-clique***, also called an ***independent set***, is a subgraph in which every two nodes are not connected by an edge. Show that every graph with n nodes contains either a clique or an anti-clique with at least $\frac{1}{2}\log_2 n$ nodes.

[A]**0.14** Use Theorem 0.25 to derive a formula for calculating the size of the monthly payment for a mortgage in terms of the principal P, interest rate I, and the number of payments t. Assume that, after t payments have been made, the loan amount is reduced to 0. Use the formula to calculate the dollar amount of each monthly payment for a 30-year mortgage with 360 monthly payments on an initial loan amount of \$100,000 with a 5% annual interest rate.

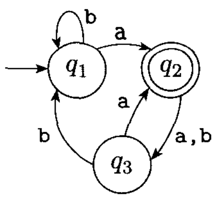
SELECTED SOLUTIONS

0.13 Make space for two piles of nodes, A and B. Then, starting with the entire graph, repeatedly add each remaining node x to A if its degree is greater than one half the number of remaining nodes and to B otherwise, and discard all nodes to which x isn't (is) connected if it was added to A (B). Continue until no nodes are left. At most half of the nodes are discarded at each of these steps, so at least $\log_2 n$ steps will occur before the process terminates. Each step adds a node to one of the piles, so one of the piles ends up with at least $\frac{1}{2}\log_2 n$ nodes. The A pile contains the nodes of a clique and the B pile contains the nodes of an anti-clique.

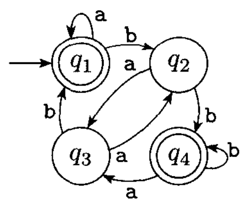
0.14 We let $P_t = 0$ and solve for Y to get the formula: $Y = PM^t(M-1)/(M^t-1)$. For $P = \$100,000$, $I = 0.05$, and $t = 360$ we have $M = 1 + (0.05)/12$. We use a calculator to find that $Y \approx \$536.82$ is the monthly payment.

EXERCISES

[A]**1.1** The following are the state diagrams of two DFAs, M_1 and M_2. Answer the following questions about each of these machines.



M_1



M_2

a. What is the start state?
b. What is the set of accept states?
c. What sequence of states does the machine go through on input aabb?
d. Does the machine accept the string aabb?
e. Does the machine accept the string ε?

[A]**1.2** Give the formal description of the machines M_1 and M_2 pictured in Exercise 1.1.

1.3 The formal description of a DFA M is $\big(\{q_1, q_2, q_3, q_4, q_5\}, \{\mathtt{u}, \mathtt{d}\}, \delta, q_3, \{q_3\}\big)$, where δ is given by the following table. Give the state diagram of this machine.

	u	d
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_2	q_4
q_4	q_3	q_5
q_5	q_4	q_5

1.4 Each of the following languages is the intersection of two simpler languages. In each part, construct DFAs for the simpler languages, then combine them using the construction discussed in footnote 3 (page 46) to give the state diagram of a DFA for the language given. In all parts $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

a. $\{w \mid w$ has at least three a's and at least two b's$\}$
[A]**b.** $\{w \mid w$ has at exactly two a's and at least two b's$\}$
c. $\{w \mid w$ has an even number of a's and one or two b's$\}$
[A]**d.** $\{w \mid w$ has an even number of a's and each a is followed by at least one b$\}$
e. $\{w \mid w$ has an even number of a's and one or two b's$\}$
f. $\{w \mid w$ has an odd number of a's and ends with a b$\}$
g. $\{w \mid w$ has even length and an odd number of a's$\}$

1.5 Each of the following languages is the complement of a simpler language. In each part, construct a DFA for the simpler language, then use it to give the state diagram of a DFA for the language given. In all parts $\Sigma = \{a, b\}$.

- [A]**a.** $\{w \mid w$ does not contain the substring ab$\}$
- [A]**b.** $\{w \mid w$ does not contain the substring baba$\}$
- **c.** $\{w \mid w$ contains neither the substrings ab nor ba$\}$
- **d.** $\{w \mid w$ is any string not in $a^*b^*\}$
- **e.** $\{w \mid w$ is any string not in $(ab^+)^*\}$
- **f.** $\{w \mid w$ is any string not in $a^* \cup b^*\}$
- **g.** $\{w \mid w$ is any string that doesn't contain exactly two a's$\}$
- **h.** $\{w \mid w$ is any string except a and b$\}$

1.6 Give state diagrams of DFAs recognizing the following languages. In all parts the alphabet is $\{0,1\}$

- **a.** $\{w \mid w$ begins with a 1 and ends with a 0$\}$
- **b.** $\{w \mid w$ contains at least three 1s$\}$
- **c.** $\{w \mid w$ contains the substring 0101, i.e., $w = x0101y$ for some x and $y\}$
- **d.** $\{w \mid w$ has length at least 3 and its third symbol is a 0$\}$
- **e.** $\{w \mid w$ starts with 0 and has odd length, or starts with 1 and has even length$\}$
- **f.** $\{w \mid w$ doesn't contain the substring 110$\}$
- **g.** $\{w \mid$ the length of w is at most 5$\}$
- **h.** $\{w \mid w$ is any string except 11 and 111$\}$
- **i.** $\{w \mid$ every odd position of w is a 1$\}$
- **j.** $\{w \mid w$ contains at least two 0s and at most one 1$\}$
- **k.** $\{\varepsilon, 0\}$
- **l.** $\{w \mid w$ contains an even number of 0s, or contains exactly two 1s$\}$
- **m.** The empty set
- **n.** All strings except the empty string

1.7 Give state diagrams of NFAs with the specified number of states recognizing each of the following languages. In all parts the alphabet is $\{0,1\}$.

- [A]**a.** The language $\{w \mid w$ ends with 00$\}$ with three states
- **b.** The language of Exercise 1.6c with five states
- **c.** The language of Exercise 1.6l with six states
- **d.** The language $\{0\}$ with two states
- **e.** The language $0^*1^*0^+$ with three states
- [A]**f.** The language $1^*(001^+)^*$ with three states
- **g.** The language $\{\varepsilon\}$ with one state
- **h.** The language 0^* with one state

1.8 Use the construction given in the proof of Theorem 1.45 to give the state diagrams of NFAs recognizing the union of the languages described in

- **a.** Exercises 1.6a and 1.6b.
- **b.** Exercises 1.6c and 1.6f.

1.9 Use the construction given in the proof of Theorem 1.47 to give the state diagrams of NFAs recognizing the concatenation of the languages described in

a. Exercises 1.6g and 1.6i.
b. Exercises 1.6b and 1.6m.

1.10 Use the construction given in the proof of Theorem 1.49 to give the state diagrams of NFAs recognizing the star of the language described in

a. Exercise 1.6b.
b. Exercise 1.6j.
c. Exercise 1.6m.

[A]**1.11** Prove that every NFA can be converted to an equivalent one that has a single accept state.

1.12 Let $D = \{w|\ w$ contains an even number of a's and an odd number of b's and does not contain the substring ab$\}$. Give a DFA with five states that recognizes D and a regular expression that generates D. (Suggestion: Describe D more simply.)

1.13 Let F be the language of all strings over $\{0,1\}$ that do not contain a pair of 1s that are separated by an odd number of symbols. Give the state diagram of a DFA with 5 states that recognizes F. (You may find it helpful first to find a 4-state NFA for the complement of F.)

1.14 **a.** Show that, if M is a DFA that recognizes language B, swapping the accept and nonaccept states in M yields a new DFA that recognizes the complement of B. Conclude that the class of regular languages is closed under complement.

b. Show by giving an example that, if M is an NFA that recognizes language C, swapping the accept and nonaccept states in M doesn't necessarily yield a new NFA that recognizes the complement of C. Is the class of languages recognized by NFAs closed under complement? Explain your answer.

1.15 Give a counterexample to show that the following construction fails to prove Theorem 1.49, the closure of the class of regular languages under the star operation.[8] Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1. Construct $N = (Q_1, \Sigma, \delta, q_1, F)$ as follows. N is supposed to recognize A_1^*.

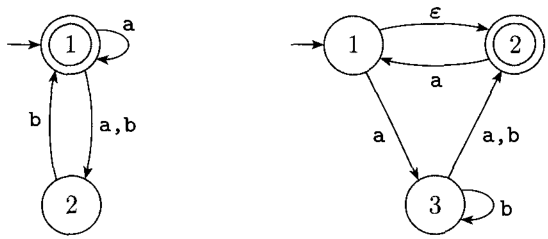
a. The states of N are the states of N_1.
b. The start state of N is the same as the start state of N_1.
c. $F = \{q_1\} \cup F_1$.
The accept states F are the old accept states plus its start state.
d. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \notin F_1 \text{ or } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1 \text{ and } a = \varepsilon. \end{cases}$$

(Suggestion: Show this construction graphically, as in Figure 1.50.)

[8]In other words, you must present a finite automaton, N_1, for which the constructed automaton N does not recognize the star of N_1's language.

1.16 Use the construction given in Theorem 1.39 to convert the following two nondeterministic finite automata to equivalent deterministic finite automata.



(a) (b)

1.17 **a.** Give an NFA recognizing the language $(01 \cup 001 \cup 010)^*$.
b. Convert this NFA to an equivalent DFA. Give only the portion of the DFA that is reachable from the start state.

1.18 Give regular expressions generating the languages of Exercise 1.6.

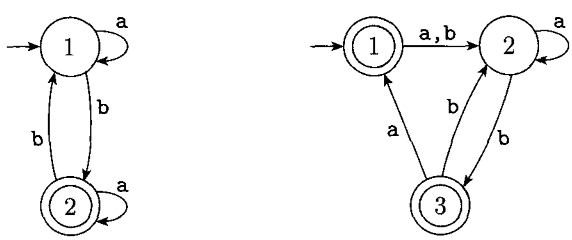
1.19 Use the procedure described in Lemma 1.55 to convert the following regular expressions to nondeterministic finite automata.

a. $(0 \cup 1)^*000(0 \cup 1)^*$
b. $(((00)^*(11)) \cup 01)^*$
c. $\emptyset^*$

1.20 For each of the following languages, give two strings that are members and two strings that are *not* members—a total of four strings for each part. Assume the alphabet $\Sigma = \{a,b\}$ in all parts.

a. a^*b^*
b. $a(ba)^*b$
c. $a^* \cup b^*$
d. $(aaa)^*$
e. $\Sigma^*a\Sigma^*b\Sigma^*a\Sigma^*$
f. $aba \cup bab$
g. $(\varepsilon \cup a)b$
h. $(a \cup ba \cup bb)\Sigma^*$

1.21 Use the procedure described in Lemma 1.60 to convert the following finite automata to regular expressions.



(a) (b)

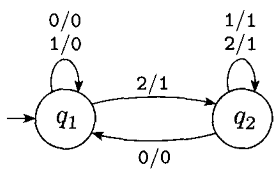
1.22 In certain programming languages, comments appear between delimiters such as /# and #/. Let C be the language of all valid delimited comment strings. A member of C must begin with /# and end with #/ but have no intervening #/. For simplicity, we'll say that the comments themselves are written with only the symbols a and b; hence the alphabet of C is $\Sigma = \{a, b, /, \#\}$.

a. Give a DFA that recognizes C.

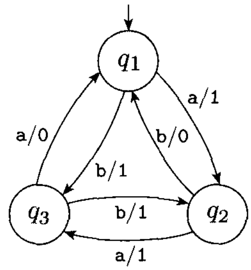
b. Give a regular expression that generates C.

^{A}**1.23** Let B be any language over the alphabet Σ. Prove that $B = B^+$ iff $BB \subseteq B$.

1.24 A ***finite state transducer*** (FST) is a type of deterministic finite automaton whose output is a string and not just *accept* or *reject*. The following are state diagrams of finite state transducers T_1 and T_2.



T_1



T_2

Each transition of an FST is labeled with two symbols, one designating the input symbol for that transition and the other designating the output symbol. The two symbols are written with a slash, /, separating them. In T_1, the transition from q_1 to q_2 has input symbol 2 and output symbol 1. Some transitions may have multiple input–output pairs, such as the transition in T_1 from q_1 to itself. When an FST computes on an input string w, it takes the input symbols $w_1 \cdots w_n$ one by one and, starting at the start state, follows the transitions by matching the input labels with the sequence of symbols $w_1 \cdots w_n = w$. Every time it goes along a transition, it outputs the corresponding output symbol. For example, on input 2212011, machine T_1 enters the sequence of states $q_1, q_2, q_2, q_2, q_2, q_1, q_1, q_1$ and produces output 1111000. On input abbb, T_2 outputs 1011. Give the sequence of states entered and the output produced in each of the following parts.

a. T_1 on input 011

b. T_1 on input 211

c. T_1 on input 121

d. T_1 on input 0202

e. T_2 on input b

f. T_2 on input bbab

g. T_2 on input bbbbbb

h. T_2 on input ε

1.25 Read the informal definition of the finite state transducer given in Exercise 1.24. Give a formal definition of this model, following the pattern in Definition 1.5 (page 35). Assume that an FST has an input alphabet Σ and an output alphabet Γ but not a set of accept states. Include a formal definition of the computation of an FST. (Hint: An FST is a 5-tuple. Its transition function is of the form $\delta\colon Q \times \Sigma \longrightarrow Q \times \Gamma$.)

1.26 Using the solution you gave to Exercise 1.25, give a formal description of the machines T_1 and T_2 depicted in Exercise 1.24.

1.27 Read the informal definition of the finite state transducer given in Exercise 1.24. Give the state diagram of an FST with the following behavior. Its input and output alphabets are {0,1}. Its output string is identical to the input string on the even positions but inverted on the odd positions. For example, on input 0000111 it should output 1010010.

1.28 Convert the following regular expressions to NFAs using the procedure given in Theorem 1.54. In all parts $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

a. $\mathtt{a}(\mathtt{abb})^* \cup \mathtt{b}$
b. $\mathtt{a}^+ \cup (\mathtt{ab})^+$
c. $(\mathtt{a} \cup \mathtt{b}^+)\mathtt{a}^+\mathtt{b}^+$

1.29 Use the pumping lemma to show that the following languages are not regular.

^{A}**a.** $A_1 = \{\mathtt{0}^n\mathtt{1}^n\mathtt{2}^n \mid n \geq 0\}$
b. $A_2 = \{www \mid w \in \{\mathtt{a}, \mathtt{b}\}^*\}$
^{A}**c.** $A_3 = \{\mathtt{a}^{2^n} \mid n \geq 0\}$ (Here, $\mathtt{a}^{2^n}$ means a string of 2^n a's.)

1.30 Describe the error in the following "proof" that $\mathtt{0}^*\mathtt{1}^*$ is not a regular language. (An error must exist because $\mathtt{0}^*\mathtt{1}^*$ *is* regular.) The proof is by contradiction. Assume that $\mathtt{0}^*\mathtt{1}^*$ is regular. Let p be the pumping length for $\mathtt{0}^*\mathtt{1}^*$ given by the pumping lemma. Choose s to be the string $\mathtt{0}^p\mathtt{1}^p$. You know that s is a member of $\mathtt{0}^*\mathtt{1}^*$, but Example 1.73 shows that s cannot be pumped. Thus you have a contradiction. So $\mathtt{0}^*\mathtt{1}^*$ is not regular.

PROBLEMS

1.31 For any string $w = w_1w_2\cdots w_n$, the ***reverse*** of w, written $w^{\mathcal{R}}$, is the string w in reverse order, $w_n \cdots w_2w_1$. For any language A, let $A^{\mathcal{R}} = \{w^{\mathcal{R}} \mid w \in A\}$.
Show that if A is regular, so is $A^{\mathcal{R}}$.

1.32 Let

$$\Sigma_3 = \left\{ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \ldots, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right\}.$$

Σ_3 contains all size 3 columns of 0s and 1s. A string of symbols in Σ_3 gives three rows of 0s and 1s. Consider each row to be a binary number and let

$$B = \{w \in \Sigma_3^* \mid \text{the bottom row of } w \text{ is the sum of the top two rows}\}.$$

For example,

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \in B, \qquad \text{but} \qquad \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \notin B.$$

Show that B is regular. (Hint: Working with $B^{\mathcal{R}}$ is easier. You may assume the result claimed in Problem 1.31.)

1.33 Let

$$\Sigma_2 = \left\{ \left[\begin{smallmatrix}0\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right], \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}1\\1\end{smallmatrix}\right] \right\}.$$

Here, Σ_2 contains all columns of 0s and 1s of height two. A string of symbols in Σ_2 gives two rows of 0s and 1s. Consider each row to be a binary number and let

$$C = \{w \in \Sigma_2^* |\ \text{the bottom row of } w \text{ is three times the top row}\}.$$

For example, $\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]\left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right] \in C$, but $\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right] \notin C$. Show that C is regular. (You may assume the result claimed in Problem 1.31.)

1.34 Let Σ_2 be the same as in Problem 1.33. Consider each row to be a binary number and let

$$D = \{w \in \Sigma_2^* |\ \text{the top row of } w \text{ is a larger number than is the bottom row}\}.$$

For example, $\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]\left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right] \in D$, but $\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]\left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right] \notin D$. Show that D is regular.

1.35 Let Σ_2 be the same as in Problem 1.33. Consider the top and bottom rows to be strings of 0s and 1s and let

$$E = \{w \in \Sigma_2^* |\ \text{the bottom row of } w \text{ is the reverse of the top row of } w\}.$$

Show that E is not regular.

1.36 Let $B_n = \{\mathtt{a}^k |\ \text{where } k \text{ is a multiple of } n\}$. Show that for each $n \geq 1$, the language B_n is regular.

1.37 Let $C_n = \{x |\ x \text{ is a binary number that is a multiple of } n\}$. Show that for each $n \geq 1$, the language C_n is regular.

1.38 An ***all*-NFA** M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ that accepts $x \in \Sigma^*$ if *every* possible state that M could be in after reading input x is a state from F. Note, in contrast, that an ordinary NFA accepts a string if *some* state among these possible states is an accept state. Prove that all-NFAs recognize the class of regular languages.

1.39 The construction in Theorem 1.54 shows that every GNFA is equivalent to a GNFA with only two states. We can show that an opposite phenomenon occurs for DFAs. Prove that for every $k > 1$ a language $A_k \subseteq \{0,1\}^*$ exists that is recognized by a DFA with k states but not by one with only $k-1$ states.

1.40 Say that string x is a ***prefix*** of string y if a string z exists where $xz = y$ and that x is a ***proper prefix*** of y if in addition $x \neq y$. In each of the following parts we define an operation on a language A. Show that the class of regular languages is closed under that operation.

[A]**a.** $NOPREFIX(A) = \{w \in A |\ \text{no proper prefix of } w \text{ is a member of } A\}$.

b. $NOEXTEND(A) = \{w \in A |\ w \text{ is not the proper prefix of any string in } A\}$.

1.41 For languages A and B, let the ***perfect shuffle*** of A and B be the language

$$\{w |\ w = a_1 b_1 \cdots a_k b_k,\ \text{where } a_1 \cdots a_k \in A \text{ and } b_1 \cdots b_k \in B,\ \text{each } a_i, b_i \in \Sigma\}.$$

Show that the class of regular languages is closed under perfect shuffle.

1.42 For languages A and B, let the ***shuffle*** of A and B be the language

$$\{w |\ w = a_1 b_1 \cdots a_k b_k,\ \text{where } a_1 \cdots a_k \in A \text{ and } b_1 \cdots b_k \in B,\ \text{each } a_i, b_i \in \Sigma^*\}.$$

Show that the class of regular languages is closed under shuffle.

1.43 Let A be any language. Define $DROP\text{-}OUT(A)$ to be the language containing all strings that can be obtained by removing one symbol from a string in A. Thus, $DROP\text{-}OUT(A) = \{xz \mid xyz \in A \text{ where } x, z \in \Sigma^*, y \in \Sigma\}$. Show that the class of regular languages is closed under the DROP-OUT operation. Give both a proof by picture and a more formal proof by construction as in Theorem 1.47.

A1.44 Let B and C be languages over $\Sigma = \{0, 1\}$. Define

$$B \overset{1}{\leftarrow} C = \{w \in B \mid \text{ for some } y \in C, \text{ strings } w \text{ and } y \text{ contain equal numbers of 1s}\}.$$

Show that the class of regular languages is closed under the $\overset{1}{\leftarrow}$ operation.

***1.45** Let $A/B = \{w \mid wx \in A \text{ for some } x \in B\}$. Show that if A is regular and B is any language then A/B is regular.

1.46 Prove that the following languages are not regular. You may use the pumping lemma and the closure of the class of regular languages under union, intersection, and complement.

a. $\{0^n 1^m 0^n \mid m, n \geq 0\}$

Ab. $\{0^m 1^n \mid m \neq n\}$

c. $\{w \mid w \in \{0,1\}^* \text{ is not a palindrome}\}$[9]

d. $\{wtw \mid w, t \in \{0,1\}^+\}$

1.47 Let $\Sigma = \{1, \#\}$ and let

$$Y = \{w \mid w = x_1\#x_2\#\cdots\#x_k \text{ for } k \geq 0, \text{ each } x_i \in 1^*, \text{ and } x_i \neq x_j \text{ for } i \neq j\}.$$

Prove that Y is not regular.

1.48 Let $\Sigma = \{0,1\}$ and let

$$D = \{w \mid w \text{ contains an equal number of occurrences of the substrings 01 and 10}\}.$$

Thus 101 $\in D$ because 101 contains a single 01 and a single 10, but 1010 $\notin D$ because 1010 contains two 10s and one 01. Show that D is a regular language.

1.49 **a.** Let $B = \{1^k y \mid y \in \{0, 1\}^* \text{ and } y \text{ contains at least } k \text{ 1s, for } k \geq 1\}$. Show that B is a regular language.

b. Let $C = \{1^k y \mid y \in \{0, 1\}^* \text{ and } y \text{ contains at most } k \text{ 1s, for } k \geq 1\}$. Show that C isn't a regular language.

A1.50 Read the informal definition of the finite state transducer given in Exercise 1.24. Prove that no FST can output $w^{\mathcal{R}}$ for every input w if the input and output alphabets are $\{0,1\}$.

1.51 Let x and y be strings and let L be any language. We say that x and y are ***distinguishable by L*** if some string z exists whereby exactly one of the strings xz and yz is a member of L; otherwise, for every string z, we have $xz \in L$ whenever $yz \in L$ and we say that x and y are ***indistinguishable by L***. If x and y are indistinguishable by L we write $x \equiv_L y$. Show that $\equiv_L$ is an equivalence relation.

---

[9]A ***palindrome*** is a string that reads the same forward and backward.

[A*]**1.52** **Myhill–Nerode theorem.** Refer to Problem 1.51. Let L be a language and let X be a set of strings. Say that X is ***pairwise distinguishable by L*** if every two distinct strings in X are distinguishable by L. Define the ***index of L*** to be the maximum number of elements in any set that is pairwise distinguishable by L. The index of L may be finite or infinite.

a. Show that, if L is recognized by a DFA with k states, L has index at most k.

b. Show that, if the index of L is a finite number k, it is recognized by a DFA with k states.

c. Conclude that L is regular iff it has finite index. Moreover, its index is the size of the smallest DFA recognizing it.

1.53 Let $\Sigma = \{0, 1, +, =\}$ and

$$ADD = \{x\texttt{=}y\texttt{+}z \mid x, y, z \text{ are binary integers, and } x \text{ is the sum of } y \text{ and } z\}.$$

Show that ADD is not regular.

1.54 Consider the language $F = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i, j, k \geq 0 \text{ and if } i = 1 \text{ then } j = k\}$.

a. Show that F is not regular.

b. Show that F acts like a regular language in the pumping lemma. In other words, give a pumping length p and demonstrate that F satisfies the three conditions of the pumping lemma for this value of p.

c. Explain why parts (a) and (b) do not contradict the pumping lemma.

1.55 The pumping lemma says that every regular language has a pumping length p, such that every string in the language can be pumped if it has length p or more. If p is a pumping length for language A, so is any length $p' \geq p$. The ***minimum pumping length*** for A is the smallest p that is a pumping length for A. For example, if $A = 01^*$, the minimum pumping length is 2. The reason is that the string $s = 0$ is in A and has length 1 yet s cannot be pumped, but any string in A of length 2 or more contains a 1 and hence can be pumped by dividing it so that $x = 0$, $y = 1$, and z is the rest. For each of the following languages, give the minimum pumping length and justify your answer.

[A]**a.** 0001^*

[A]**b.** 0^*1^*

c. $001 \cup 0^*1^*$

[A]**d.** $0^*1^+0^+1^* \cup 10^*1$

e. $(01)^*$

f. ε

g. $1^*01^*01^*$

h. $10(11^*0)^*0$

i. 1011

j. Σ^*

[*]**1.56** If A is a set of natural numbers and k is a natural number greater than 1, let

$$B_k(A) = \{w \mid w \text{ is the representation in base } k \text{ of some number in } A\}.$$

Here, we do not allow leading 0s in the representation of a number. For example, $B_2(\{3, 5\}) = \{11, 101\}$ and $B_3(\{3, 5\}) = \{10, 12\}$. Give an example of a set A for which $B_2(A)$ is regular but $B_3(A)$ is not regular. Prove that your example works.

***1.57** If A is any language, let $A_{\frac{1}{2}-}$ be the set of all first halves of strings in A so that

$$A_{\frac{1}{2}-} = \{x \mid \text{ for some } y,\ |x| = |y| \text{ and } xy \in A\}.$$

Show that, if A is regular, then so is $A_{\frac{1}{2}-}$.

***1.58** If A is any language, let $A_{\frac{1}{3}-\frac{1}{3}}$ be the set of all strings in A with their middle thirds removed so that

$$A_{\frac{1}{3}-\frac{1}{3}} = \{xz \mid \text{ for some } y,\ |x| = |y| = |z| \text{ and } xyz \in A\}.$$

Show that, if A is regular, then $A_{\frac{1}{3}-\frac{1}{3}}$ is not necessarily regular.

1.59** Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let h be a state of M called its "home". A ***synchronizing sequence for M and h is a string $s \in \Sigma^*$ where $\delta(q, s) = h$ for every $q \in Q$. (Here we have extended δ to strings, so that $\delta(q, s)$ equals the state where M ends up when M starts at state q and reads input s.) Say that M is ***synchronizable*** if it has a synchronizing sequence for some state h. Prove that, if M is a k-state synchronizable DFA, then it has a synchronizing sequence of length at most k^3. Can you improve upon this bound?

1.60 Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. For each $k \geq 1$, let C_k be the language consisting of all strings that contain an a exactly k places from the right-hand end. Thus $C_k = \Sigma^*\mathtt{a}\Sigma^{k-1}$. Describe an NFA with $k + 1$ states that recognizes C_k, both in terms of a state diagram and a formal description.

1.61 Consider the languages C_k defined in Problem 1.60. Prove that for each k, no DFA can recognize C_k with fewer than 2^k states.

1.62 Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. For each $k \geq 1$, let D_k be the language consisting of all strings that have at least one a among the last k symbols. Thus $D_k = \Sigma^*\mathtt{a}(\Sigma \cup \varepsilon)^{k-1}$. Describe an DFA with at most $k + 1$ states that recognizes D_k, both in terms of a state diagram and a formal description.

1.63
- **a.** Let A be an infinite regular language. Prove that A can be split into two infinite disjoint regular subsets.
- **b.** Let B and D be two languages. Write $B \Subset D$ if $B \subseteq D$ and D contains infinitely many strings that are not in B. Show that, if B and D are two regular languages where $B \Subset D$, then we can find a regular language C where $B \Subset C \Subset D$.

1.64 Let N be an NFA with k states that recognizes some language A.

- **a.** Show that, if A is nonempty, A contains some string of length at most k.
- **b.** Show that, by giving an example, that part (a) is not necessarily true if you replace both A's by $\overline{A}$.
- **c.** Show that, if $\overline{A}$ is nonempty, $\overline{A}$ contains some string of length at most 2^k.
- **d.** Show that the bound given in part (b) is nearly tight; that is, for each k, demonstrate an NFA recognizing a language A_k where $\overline{A_k}$ is nonempty and where $\overline{A_k}$'s shortest member strings are of length exponential in k. Come as close to the bound in (b) as you can.

*1.65 Prove that, for each $n > 0$, a language B_n exists where

a. B_n is recognizable by a NFA that has n states, and

b. if $B_n = A_1 \cup \cdots \cup A_k$, for regular languages A_i, then at least one of the A_i requires a DFA with exponentially many states.

SELECTED SOLUTIONS

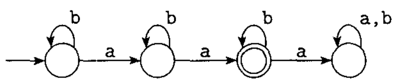
1.1 For M_1: **(a)** q_1; **(b)** $\{q_2\}$; **(c)** q_1, q_2, q_3, q_1, q_1; **(d)** No; **(e)** No
For M_2: **(a)** q_1; **(b)** $\{q_1, q_4\}$; **(c)** q_1, q_1, q_1, q_2, q_4; **(d)** Yes; **(e)** Yes

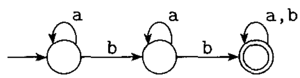
1.2 $M_2 = (\{q_1, q_2, q_3\}, \{\mathtt{a}, \mathtt{b}\}, \delta_1, q_1, \{q_2\})$.
$M_3 = (\{q_1, q_2, q_3, q_4\}, \{\mathtt{a}, \mathtt{b}\}, \delta_2, q_1, \{q_1, q_4\})$.
The transition functions are

δ_1	a	b
q_1	q_2	q_1
q_2	q_3	q_3
q_3	q_2	q_1

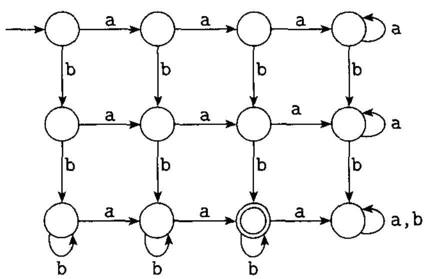
δ_2	a	b
q_1	q_1	q_2
q_2	q_3	q_4
q_3	q_2	q_1
q_4	q_3	q_4

1.4 **(b)** The following are DFAs for the two languages $\{w \mid w \text{ has exactly three a's}\}$ and $\{w \mid w \text{ has at least two b's}\}$:

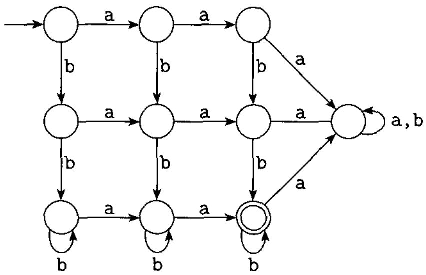




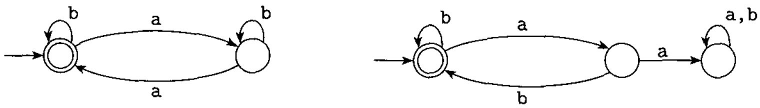
Combining them using the intersection construction gives the DFA:



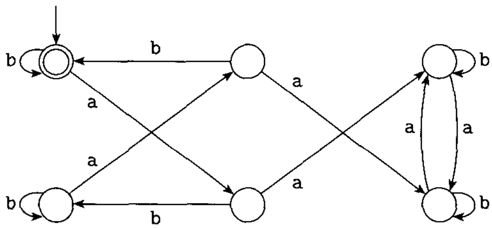
Though the problem doesn't request you to simplify the DFA, certain states can be combined to give



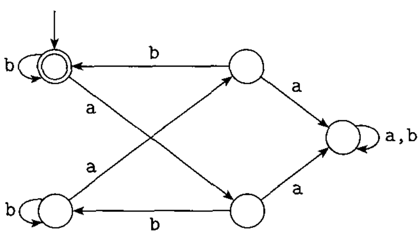
(d) These are DFAs for the two languages $\{w|\ w$ has an even number of a's$\}$ and $\{w|$ each a is followed by at least one b$\}$:



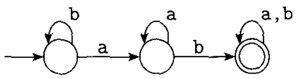
Combining them using the intersection construction gives the DFA:

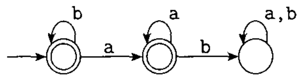


Though the problem doesn't request you to simplify the DFA, certain states can be combined to give

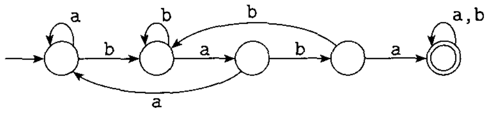


1.5 **(a)** The left-hand DFA recognizes $\{w|\ w \text{ contains } \texttt{ab}\}$. The right-hand DFA recognizes its complement, $\{w|\ w \text{ doesn't contain } \texttt{ab}\}$.

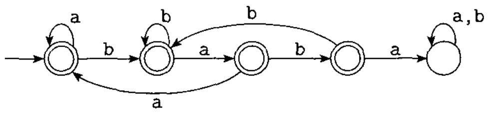




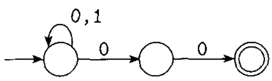
(b) This DFA recognizes $\{w|\ w \text{ contains } \texttt{baba}\}$.



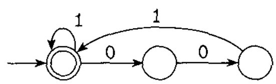
This DFA recognizes $\{w|\ w \text{ does not contain } \texttt{baba}\}$.



1.7 **(a)**



(f)



1.11 Let $N = (Q, \Sigma, \delta, q_0, F)$ be any NFA. Construct an NFA N' with a single accept state that accepts the same language as N. Informally, N' is exactly like N except it has ε-transitions from the states corresponding to the accept states of N, to a new accept state, q_{accept}. State q_{accept} has no emerging transitions. More formally, $N' = (Q \cup \{q_{\text{accept}}\}, \Sigma, \delta', q_0, \{q_{\text{accept}}\})$, where for each $q \in Q$ and $a \in \Sigma$

$$\delta'(q,a) = \begin{cases} \delta(q,a) & \text{if } a \neq \varepsilon \text{ or } q \notin F \\ \delta(q,a) \cup \{q_{\text{accept}}\} & \text{if } a = \varepsilon \text{ and } q \in F \end{cases}$$

and $\delta'(q_{\text{accept}}, a) = \emptyset$ for each $a \in \Sigma_\varepsilon$.

1.23 We prove both directions of the "iff."
($\rightarrow$) Assume that $B = B^+$ and show that $BB \subseteq B$.
For every language $BB \subseteq B^+$ holds, so if $B = B^+$, then $BB \subseteq B$.
($\leftarrow$) Assume that $BB \subseteq B$ and show that $B = B^+$.
For every language $B \subseteq B^+$, so we need to show only $B^+ \subseteq B$. If $w \in B^+$, then $w = x_1x_2\cdots x_k$ where each $x_i \in B$ and $k \geq 1$. Because $x_1, x_2 \in B$ and $BB \subseteq B$, we have $x_1x_2 \in B$. Similarly, because x_1x_2 is in B and x_3 is in B, we have $x_1x_2x_3 \in B$. Continuing in this way, $x_1 \cdots x_k \in B$. Hence $w \in B$, and so we may conclude that $B^+ \subseteq B$.

The latter argument may be written formally as the following proof by induction. Assume that $BB \subseteq B$.
Claim: For each $k \geq 1$, if $x_1, \ldots, x_k \in B$, then $x_1 \cdots x_k \in B$.
Basis: Prove for $k = 1$. This statement is obviously true.
Induction step: For each $k \geq 1$, assume that the claim is true for k and prove it to be true for $k + 1$.
If $x_1, \ldots, x_k, x_{k+1} \in B$, then by the induction assumption, $x_1 \cdots x_k \in B$. Therefore $x_1 \cdots x_k x_{k+1} \in BB$, but $BB \subseteq B$, so $x_1 \cdots x_{k+1} \in B$. That proves the induction step and the claim. The claim implies that, if $BB \subseteq B$, then $B^+ \subseteq B$.

1.29 **(a)** Assume that $A_1 = \{0^n1^n2^n \mid n \geq 0\}$ is regular. Let p be the pumping length given by the pumping lemma. Choose s to be the string $0^p1^p2^p$. Because s is a member of A_1 and s is longer than p, the pumping lemma guarantees that s can be split into three pieces, $s = xyz$, where for any $i \geq 0$ the string xy^iz is in A_1. Consider two possibilities:

1. The string y consists only of 0s, only of 1s, or only of 2s. In these cases the string $xyyz$ will not have equal numbers of 0s, 1s, and 2s. Hence $xyyz$ is not a member of A_1, a contradiction.
2. The string y consists of more than one kind of symbol. In this case $xyyz$ will have the 0s, 1s, or 2s out of order. Hence $xyyz$ is not a member of A_1, a contradiction.

Either way we arrive at a contradiction. Therefore, A_1 is not regular.

(c) Assume that $A_3 = \{a^{2^n} \mid n \geq 0\}$ is regular. Let p be the pumping length given by the pumping lemma. Choose s to be the string a^{2^p}. Because s is a member of A_1 and s is longer than p, the pumping lemma guarantees that s can be split into three pieces, $s = xyz$, satisfying the three conditions of the pumping lemma.

The third condition tells us that $|xy| \leq p$. Furthermore, $p < 2^p$ and so $|y| < 2^p$. Therefore $|xyyz| = |xyz| + |y| < 2^p + 2^p = 2^{p+1}$. The second condition requires $|y| > 1$ so $2^p < |xyyz| < 2^{p+1}$. The length of $xyyz$ cannot be a power of 2. Hence $xyyz$ is not a member of A_3, a contradiction. Therefore, A_3 is not regular.

1.40 Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA recognizing A, where A is some regular language. Construct $M' = (Q', \Sigma, \delta', q_0', F')$ recognizing $NOPREFIX(A)$ as follows:

1. $Q' = Q$.
2. For $r \in Q'$ and $a \in \Sigma$ define $\delta'(r, a) = \begin{cases} \delta(r, a) & \text{if } r \notin F \\ \emptyset & \text{if } r \in F. \end{cases}$
3. $q_0' = q_0$.
4. $F' = F$.

1.44 Let $M_B = (Q_B, \Sigma, \delta_B, q_B, F_B)$ and $M_C = (Q_C, \Sigma, \delta_C, q_C, F_C)$ be DFAs recognizing B and C respectively. Construct NFA $M = (Q, \Sigma, \delta, q_0, F)$ that recognizes $B \stackrel{1}{\leftarrow} C$ as follows. To decide whether its input w is in $B \stackrel{1}{\leftarrow} C$, the machine M checks that $w \in B$, and in parallel, nondeterministically guesses a string y that contains the same number of 1s as contained in w and checks that $y \in C$.

1. $Q = Q_B \times Q_C$.
2. For $(q, r) \in Q$ and $a \in \Sigma$ define

$$\delta((q,r),a) = \begin{cases} \{(\delta_B(q,0), r)\} & \text{if } a = 0 \\ \{(\delta_B(q,1), \delta_C(r,1))\} & \text{if } a = 1 \\ \{(q, \delta_C(r,0))\} & \text{if } a = \varepsilon. \end{cases}$$

3. $q_0 = (q_B, q_C)$.
4. $F = F_B \times F_C$.

1.46 **(b)** Let $B = \{0^m 1^n \mid m \neq n\}$. Observe that $\overline{B} \cap 0^*1^* = \{0^k 1^k \mid k \geq 0\}$. If B were regular, then $\overline{B}$ would be regular and so would $\overline{B} \cap 0^*1^*$. But we already know that $\{0^k 1^k \mid k \geq 0\}$ isn't regular, so B cannot be regular.

Alternatively, we can prove B to be nonregular by using the pumping lemma directly, though doing so is trickier. Assume that $B = \{0^m 1^n \mid m \neq n\}$ is regular. Let p be the pumping length given by the pumping lemma. Observe that $p!$ is divisible by all integers from 1 to p, where $p! = p(p-1)(p-2)\cdots 1$. The string $s = 0^p 1^{p+p!} \in B$, and $|s| \geq p$. Thus the pumping lemma implies that s can be divided as xyz with $x = 0^a$, $y = 0^b$, and $z = 0^c 1^{p+p!}$, where $b \geq 1$ and $a+b+c = p$. Let s' be the string $xy^{i+1}z$, where $i = p!/b$. Then $y^i = 0^{p!}$ so $y^{i+1} = 0^{b+p!}$, and so $xyz = 0^{a+b+c+p!}1^{p+p!}$. That gives $xyz = 0^{p+p!}1^{p+p!} \notin B$, a contradiction.

1.50 Assume to the contrary that some FST T outputs $w^{\mathcal{R}}$ on input w. Consider the input strings 00 and 01. On input 00, T must output 00, and on input 01, T must output 10. In both cases the first input bit is a 0 but the first output bits differ. Operating in this way is impossible for an FST because it produces its first output bit before it reads its second input. Hence no such FST can exist.

1.52 **(a)** We prove this assertion by contradiction. Let M be a k-state DFA that recognizes L. Suppose for a contradiction that L has index greater than k. That means some set X with more than k elements is pairwise distinguishable by L. Because M has k states, the pigeonhole principle implies that X contains two distinct strings x and y, where $\delta(q_0, x) = \delta(q_0, y)$. Here $\delta(q_0, x)$ is the state that M is in after starting in the start state q_0 and reading input string x. Then, for any string $z \in \Sigma^*$, $\delta(q_0, xz) = \delta(q_0, yz)$. Therefore either both xz and yz are in L or neither are in L. But then x and y aren't distinguishable by L, contradicting our assumption that X is pairwise distinguishable by L.

(b) Let $X = \{s_1, \ldots, s_k\}$ be pairwise distinguishable by L. We construct DFA $M = (Q, \Sigma, \delta, q_0, F)$ with k states recognizing L. Let $Q = \{q_1, \ldots, q_k\}$, and define $\delta(q_i, a)$ to be q_j, where $s_j \equiv_L s_i a$ (the relation $\equiv_L$ is defined in Problem 1.51). Note that $s_j \equiv_L s_i a$ for some $s_j \in X$; otherwise, $X \cup s_i a$ would have $k+1$ elements and would be pairwise distinguishable by L, which would contradict the assumption that L has index k. Let $F = \{q_i \mid s_i \in L\}$. Let the start state q_0 be the q_i such that $s_i \equiv_L \varepsilon$. M is constructed so that, for any state q_i, $\{s \mid \delta(q_0, s) = q_i\} = \{s \mid s \equiv_L s_i\}$. Hence M recognizes L.

(c) Suppose that L is regular and let k be the number of states in a DFA recognizing L. Then from part (a) L has index at most k. Conversely, if L has index k, then by part (b) it is recognized by a DFA with k states and thus is regular. To show that the index of L is the size of the smallest DFA accepting it, suppose that L's index is *exactly* k. Then, by part (b), there is a k-state DFA accepting L. That is the smallest such DFA because if it were any smaller, then we could show by part (a) that the index of L is less than k.

1.55 **(a)** The minimum pumping length is 4. The string 000 is in the language but cannot be pumped, so 3 is not a pumping length for this language. If s has length 4 or more, it contains 1s. By dividing s onto xyz, where x is 000 and y is the first 1 and z is everything afterward, we satisfy the pumping lemma's three conditions.

(b) The minimum pumping length is 1. The pumping length cannot be 0 because the string ε is in the language and it cannot be pumped. Every nonempty string in the language can be divided into xyz, where $x = \varepsilon$ and y is the first character and z is the remainder. This division satisfies the three conditions.

(d) The minimum pumping length is 3. The pumping length cannot be 2 because the string 11 is in the language and it cannot be pumped. Let s be a string in the language of length at least 3. If s is generated by $0^*1^+0^+1^*$, we can write it as xyz, where x is the empty string, y is the first symbol of s, and z is the remainder of s. Breaking s up in this way shows that it can be pumped. If s is generated by 10^*1, we can write it as xyz, where $x = 1$ and $y = 0$ and z is the remainder of s. This division gives a way to pump s.

EXERCISES

2.1 Recall the CFG G_4 that we gave in Example 2.4. For convenience, let's rename its variables with single letters as follows.

$$E \to E + T \mid T$$
$$T \to T \times F \mid F$$
$$F \to (\, E\,) \mid \mathtt{a}$$

Give parse trees and derivations for each string.

a. a

b. a+a

c. a+a+a

d. ((a))

2.2 **a.** Use the languages $A = \{\mathtt{a}^m\mathtt{b}^n\mathtt{c}^n \mid m, n \geq 0\}$ and $B = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^m \mid m, n \geq 0\}$ together with Example 2.36 to show that the class of context-free languages is not closed under intersection.

b. Use part (a) and DeMorgan's law (Theorem 0.20) to show that the class of context-free languages is not closed under complementation.

A2.3 Answer each part for the following context-free grammar G.

$$R \to XRX \mid S$$
$$S \to \mathtt{a}T\mathtt{b} \mid \mathtt{b}T\mathtt{a}$$
$$T \to XTX \mid X \mid \varepsilon$$
$$X \to \mathtt{a} \mid \mathtt{b}$$

a. What are the variables of G?

b. What are the terminals of G?

c. Which is the start variable of G?

d. Give three strings in $L(G)$.

e. Give three strings *not* in $L(G)$.

f. True or False: $T \Rightarrow \mathtt{aba}$.

g. True or False: $T \overset{*}{\Rightarrow} \mathtt{aba}$.

h. True or False: $T \Rightarrow T$.

i. True or False: $T \overset{*}{\Rightarrow} T$.

j. True or False: $XXX \overset{*}{\Rightarrow} \mathtt{aba}$.

k. True or False: $X \overset{*}{\Rightarrow} \mathtt{aba}$.

l. True or False: $T \overset{*}{\Rightarrow} XX$.

m. True or False: $T \overset{*}{\Rightarrow} XXX$.

n. True or False: $S \overset{*}{\Rightarrow} \varepsilon$.

o. Give a description in English of $L(G)$.

2.4 Give context-free grammars that generate the following languages. In all parts the alphabet Σ is $\{0,1\}$.

Aa. $\{w \mid w$ contains at least three 1s$\}$

b. $\{w \mid w$ starts and ends with the same symbol$\}$

c. $\{w \mid$ the length of w is odd$\}$

Ad. $\{w \mid$ the length of w is odd and its middle symbol is a 0$\}$

e. $\{w \mid w = w^{\mathcal{R}}$, that is, w is a palindrome$\}$

f. The empty set

2.5 Give informal descriptions and state diagrams of pushdown automata for the languages in Exercise 2.4.

2.6 Give context-free grammars generating the following languages.

[A]**a.** The set of strings over the alphabet $\{\mathtt{a},\mathtt{b}\}$ with more a's than b's

b. The complement of the language $\{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$

[A]**c.** $\{w\#x \mid w^{\mathcal{R}} \text{ is a substring of } x \text{ for } w, x \in \{\mathtt{0},\mathtt{1}\}^*\}$

d. $\{x_1\#x_2\#\cdots\#x_k \mid k \geq 1, \text{ each } x_i \in \{\mathtt{a},\mathtt{b}\}^*, \text{ and for some } i \text{ and } j,\ x_i = x_j^{\mathcal{R}}\}$

[A]**2.7** Give informal English descriptions of PDAs for the languages in Exercise 2.6.

[A]**2.8** Show that the string `the girl touches the boy with the flower` has two different leftmost derivations in grammar G_2 on page 101. Describe in English the two different meanings of this sentence.

2.9 Give a context-free grammar that generates the language

$$A = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i = j \text{ or } j = k \text{ where } i, j, k \geq 0\}.$$

Is your grammar ambiguous? Why or why not?

2.10 Give an informal description of a pushdown automaton that recognizes the language A in Exercise 2.9.

2.11 Convert the CFG G_4 given in Exercise 2.1 to an equivalent PDA, using the procedure given in Theorem 2.20.

2.12 Convert the CFG G given in Exercise 2.3 to an equivalent PDA, using the procedure given in Theorem 2.20.

2.13 Let $G = (V, \Sigma, R, S)$ be the following grammar. $V = \{S, T, U\}$; $\Sigma = \{\mathtt{0}, \#\}$; and R is the set of rules:

$$\begin{aligned} S &\rightarrow TT \mid U \\ T &\rightarrow \mathtt{0}T \mid T\mathtt{0} \mid \# \\ U &\rightarrow \mathtt{0}U\mathtt{00} \mid \# \end{aligned}$$

a. Describe $L(G)$ in English.

b. Prove that $L(G)$ is not regular.

2.14 Convert the following CFG into an equivalent CFG in Chomsky normal form, using the procedure given in Theorem 2.9.

$$\begin{aligned} A &\rightarrow BAB \mid B \mid \varepsilon \\ B &\rightarrow \mathtt{00} \mid \varepsilon \end{aligned}$$

2.15 Give a counterexample to show that the following construction fails to prove that the class of context-free languages is closed under star. Let A be a CFL that is generated by the CFG $G = (V, \Sigma, R, S)$. Add the new rule $S \rightarrow SS$ and call the resulting grammar G'. This grammar is supposed to generate A^*.

2.16 Show that the class of context-free languages is closed under the regular operations, union, concatenation, and star.

2.17 Use the results of Problem 2.16 to give another proof that every regular language is context free, by showing how to convert a regular expression directly to an equivalent context-free grammar.

PROBLEMS

[A]**2.18** **a.** Let C be a context-free language and R be a regular language. Prove that the language $C \cap R$ is context free.

b. Use part (a) to show that the language $A = \{w |\ w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*$ and contains equal numbers of a's, b's, and c's$\}$ is not a CFL.

*__2.19__ Let CFG G be

$$S \rightarrow \mathtt{a}S\mathtt{b} \mid \mathtt{b}Y \mid Y\mathtt{a}$$
$$Y \rightarrow \mathtt{b}Y \mid \mathtt{a}Y \mid \varepsilon$$

Give a simple description of $L(G)$ in English. Use that description to give a CFG for $\overline{L(G)}$, the complement of $L(G)$.

2.20 Let $A/B = \{w |\ wx \in A$ for some $x \in B\}$. Show that, if A is context free and B is regular, then A/B is context free.

*__2.21__ Let $\Sigma = \{\mathtt{a},\mathtt{b}\}$. Give a CFG generating the language of strings with twice as many a's as b's. Prove that your grammar is correct.

*__2.22__ Let $C = \{x\#y |\ x, y \in \{\mathtt{0},\mathtt{1}\}^*$ and $x \neq y\}$. Show that C is a context-free language.

*__2.23__ Let $D = \{xy | x, y \in \{\mathtt{0},\mathtt{1}\}^*$ and $|x| = |y|$ but $x \neq y\}$. Show that D is a context-free language.

*__2.24__ Let $E = \{\mathtt{a}^i\mathtt{b}^j |\ i \neq j$ and $2i \neq j\}$. Show that E is a context-free language.

2.25 For any language A, let $SUFFIX(A) = \{v |\ uv \in A$ for some string $u\}$. Show that the class of context-free languages is closed under the $SUFFIX$ operation.

2.26 Show that, if G is a CFG in Chomsky normal form, then for any string $w \in L(G)$ of length $n \geq 1$, exactly $2n - 1$ steps are required for any derivation of w.

*__2.27__ Let $G = (V, \Sigma, R, \langle \text{STMT} \rangle)$ be the following grammar.

$$\langle \text{STMT} \rangle \rightarrow \langle \text{ASSIGN} \rangle \mid \langle \text{IF-THEN} \rangle \mid \langle \text{IF-THEN-ELSE} \rangle$$
$$\langle \text{IF-THEN} \rangle \rightarrow \mathtt{if\ condition\ then}\ \langle \text{STMT} \rangle$$
$$\langle \text{IF-THEN-ELSE} \rangle \rightarrow \mathtt{if\ condition\ then}\ \langle \text{STMT} \rangle\ \mathtt{else}\ \langle \text{STMT} \rangle$$
$$\langle \text{ASSIGN} \rangle \rightarrow \mathtt{a{:}{=}1}$$

$$\Sigma = \{\mathtt{if}, \mathtt{condition}, \mathtt{then}, \mathtt{else}, \mathtt{a{:}{=}1}\}.$$
$$V = \{\langle \text{STMT} \rangle, \langle \text{IF-THEN} \rangle, \langle \text{IF-THEN-ELSE} \rangle, \langle \text{ASSIGN} \rangle\}$$

G is a natural-looking grammar for a fragment of a programming language, but G is ambiguous.

a. Show that G is ambiguous.

b. Give a new unambiguous grammar for the same language.

*__2.28__ Give unambiguous CFGs for the following languages.

a. $\{w |$ in every prefix of w the number of a's is at least the number of b's$\}$

b. $\{w |$ the number of a's and b's in w are equal$\}$

c. $\{w |$ the number of a's is at least the number of b's$\}$

*__2.29__ Show that the language A in Exercise 2.9 is inherently ambiguous.

2.30 Use the pumping lemma to show that the following languages are not context free.

a. $\{0^n 1^n 0^n 1^n \mid n \geq 0\}$

[A]**b.** $\{0^n \# 0^{2n} \# 0^{3n} \mid n \geq 0\}$

[A]**c.** $\{w\#t \mid w \text{ is a substring of } t, \text{ where } w, t \in \{a, b\}^*\}$

d. $\{t_1\#t_2\# \cdots \#t_k \mid k \geq 2, \text{ each } t_i \in \{a, b\}^*, \text{ and } t_i = t_j \text{ for some } i \neq j\}$

2.31 Let B be the language of all palindromes over $\{0,1\}$ containing an equal number of 0s and 1s. Show that B is not context free.

*__2.32__ Let $\Sigma = \{1, 2, 3, 4\}$ and $C = \{w \in \Sigma^* \mid$ in w, the number of 1s equals the number of 2s, and the number of 3s equals the number of 4s$\}$. Show that C is not context free.

2.33 Show that $F = \{a^i b^j \mid i \neq kj \text{ for every positive integer } k\}$ is not context free.

2.34 Consider the language $B = L(G)$, where G is the grammar given in Exercise 2.13. The pumping lemma for context-free languages, Theorem 2.34, states the existence of a pumping length p for B. What is the minimum value of p that works in the pumping lemma? Justify your answer.

2.35 Let G be a CFG in Chomsky normal form that contains b variables. Show that, if G generates some string with a derivation having at least 2^b steps, $L(G)$ is infinite.

2.36 Give an example of a language that is not context free but that acts like a CFL in the pumping lemma. Prove that your example works. (See the analogous example for regular languages in Problem 1.54.)

*__2.37__ Prove the following stronger form of the pumping lemma, wherein *both* pieces v and y must be nonempty when the string s is broken up.

If A is a context-free language, then there is a number k where, if s is any string in A of length at least k, then s may be divided into five pieces, $s = uvxyz$, satisfying the conditions:

a. for each $i \geq 0$, $uv^i xy^i z \in A$,

b. $v \neq \varepsilon$ and $y \neq \varepsilon$, and

c. $|vxy| \leq k$.

[A]**2.38** Refer to Problem 1.41 for the definition of the perfect shuffle operation. Show that the class of context-free languages is not closed under perfect shuffle.

2.39 Refer to Problem 1.42 for the definition of the shuffle operation. Show that the class of context-free languages is not closed under shuffle.

*__2.40__ Say that a language is ***prefix-closed*** if the prefix of any string in the language is also in the language. Let C be an infinite, prefix-closed, context-free language. Show that C contains an infinite regular subset.

*__2.41__ Read the definitions of $NOPREFIX(A)$ and $NOEXTEND(A)$ in Problem 1.40.

a. Show that the class of CFLs is not closed under $NOPREFIX$ operation.

b. Show that the class of CFLs is not closed under $NOEXTEND$ operation.

2.42 Let $\Sigma = \{1, \#\}$ and $Y = \{w \mid w = t_1\#t_2\# \cdots \#t_k \text{ for } k \geq 0, \text{ each } t_i \in 1^*, \text{ and } t_i \neq t_j \text{ whenever } i \neq j\}$. Prove that Y is not context free.

2.43 For strings w and t, write $w \stackrel{\circ}{=} t$ if the symbols of w are a permutation of the symbols of t. In other words, $w \stackrel{\circ}{=} t$ if t and w have the same symbols in the same quantities, but possibly in a different order.

For any string w, define $SCRAMBLE(w) = \{t \mid t \stackrel{\circ}{=} w\}$. For any language A, let $SCRAMBLE(A) = \{t \mid t \in SCRAMBLE(w)$ for some $w \in A\}$.

a. Show that, if $\Sigma = \{0, 1\}$, then the *SCRAMBLE* of a regular language is context free.

b. What happens in part (a) if Σ contains 3 or more symbols? Prove your answer.

2.44 If A and B are languages, define $A \diamond B = \{xy \mid x \in A$ and $y \in B$ and $|x| = |y|\}$. Show that if A and B are regular languages, then $A \diamond B$ is a CFL.

***2.45** Let $A = \{wtw^{\mathcal{R}} \mid w, t \in \{0,1\}^*$ and $|w| = |t|\}$. Prove that A is not a context-free language.

SELECTED SOLUTIONS

2.3 **(a)** R, X, S, T; **(b)** a, b; **(c)** R; **(d)** Three strings in G are ab, ba, and aab; **(e)** Three strings not in G are a, b, and ε; **(f)** False; **(g)** True; **(h)** False; **(i)** True; **(j)** True; **(k)** False; **(l)** True; **(m)** True; **(n)** False; **(o)** $L(G)$ consists of all strings over a and b that are not palindromes.

2.4 **(a)** $S \to R1R1R1R$
$R \to 0R \mid 1R \mid \varepsilon$

(d) $S \to 0 \mid 0S0 \mid 0S1 \mid 1S0 \mid 1S1$

2.6 **(a)** $S \to TaT$
$T \to TT \mid aTb \mid bTa \mid a \mid \varepsilon$
T generates all strings with at least as many a's as b's, and S forces an extra a.

(c) $S \to TX$
$T \to 0T0 \mid 1T1 \mid \#X$
$X \to 0X \mid 1X \mid \varepsilon$

2.7 **(a)** The PDA uses its stack to count the number of a's minus the number of b's. It enters an accepting state whenever this count is 0. In more detail, it operates as follows. The PDA scans across the input. If it sees a b and its top stack symbol is a a, it pops the stack. Similarly, if it scans a a and its top stack symbol is a b, it pops the stack. In all other cases, it pushes the input symbol onto the stack. After the PDA scans the input, if b is on top of the stack, it accepts. Otherwise it rejects.

(c) The PDA scans across the input string and pushes every symbol it reads until it reads a #. If # is never encountered, it rejects. Then, the PDA skips over part of the input, nondeterministically deciding when to stop skipping. At that point, it compares the next input symbols with the symbols it pops off the stack. At any disagreement, or if the input finishes while the stack is nonempty, this branch of the computation rejects. If the stack becomes empty, the machine reads the rest of the input and accepts.

2.8 Here is one derivation:
⟨SENTENCE⟩ ⇒ ⟨NOUN-PHRASE⟩⟨VERB-PHRASE⟩ ⇒
⟨CMPLX-NOUN⟩⟨VERB-PHRASE⟩ ⇒
⟨CMPLX-NOUN⟩⟨CMPLX-VERB⟩⟨PREP-PHRASE⟩ ⇒
⟨ARTICLE⟩⟨NOUN⟩⟨CMPLX-VERB⟩⟨PREP-PHRASE⟩ ⇒
The boy ⟨VERB⟩⟨NOUN-PHRASE⟩⟨PREP-PHRASE⟩ ⇒
The boy ⟨VERB⟩⟨NOUN-PHRASE⟩⟨PREP⟩⟨CMPLX-NOUN⟩ ⇒
The boy touches ⟨NOUN-PHRASE⟩⟨PREP⟩⟨CMPLX-NOUN⟩ ⇒
The boy touches ⟨CMPLX-NOUN⟩⟨PREP⟩⟨CMPLX-NOUN⟩ ⇒
The boy touches ⟨ARTICLE⟩⟨NOUN⟩⟨PREP⟩⟨CMPLX-NOUN⟩ ⇒
The boy touches the girl with ⟨CMPLX-NOUN⟩ ⇒
The boy touches the girl with ⟨ARTICLE⟩⟨NOUN⟩ ⇒
The boy touches the girl with the flower

Here is another derivation:
⟨SENTENCE⟩ ⇒ ⟨NOUN-PHRASE⟩⟨VERB-PHRASE⟩ ⇒
⟨CMPLX-NOUN⟩⟨VERB-PHRASE⟩ ⇒ ⟨ARTICLE⟩⟨NOUN⟩⟨VERB-PHRASE⟩ ⇒
The boy ⟨VERB-PHRASE⟩ ⇒ The boy ⟨CMPLX-VERB⟩ ⇒
The boy ⟨VERB⟩⟨NOUN-PHRASE⟩ ⇒
The boy touches ⟨NOUN-PHRASE⟩ ⇒
The boy touches ⟨CMPLX-NOUN⟩⟨PREP-PHRASE⟩ ⇒
The boy touches ⟨ARTICLE⟩⟨NOUN⟩⟨PREP-PHRASE⟩ ⇒
The boy touches the girl ⟨PREP-PHRASE⟩ ⇒
The boy touches the girl ⟨PREP⟩⟨CMPLX-NOUN⟩ ⇒
The boy touches the girl with ⟨CMPLX-NOUN⟩ ⇒
The boy touches the girl with ⟨ARTICLE⟩⟨NOUN⟩ ⇒
The boy touches the girl with the flower

Each of these derivations corresponds to a different English meaning. In the first derivation, the sentence means that the boy used the flower to touch the girl. In the second derivation, the girl is holding the flower when the boy touches her.

2.18 **(a)** Let C be a context-free language and R be a regular language. Let P be the PDA that recognizes C, and D be the DFA that recognizes R. If Q is the set of states of P and Q' is the set of states of D, we construct a PDA P' that recognizes $C \cap R$ with the set of states $Q \times Q'$. P' will do what P does and also keep track of the states of D. It accepts a string w if and only if it stops at a state $q \in F_P \times F_D$, where F_P is the set of accept states of P and F_D is the set of accept states of D. Since $C \cap R$ is recognized by P', it is context free.

(b) Let R be the regular language $\mathtt{a}^*\mathtt{b}^*\mathtt{c}^*$. If A were a CFL then $A \cap R$ would be a CFL by part (a). However, $A \cap R = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 0\}$, and Example 2.36 proves that $A \cap R$ is not context free. Thus A is not a CFL.

2.30 **(b)** Let $B = \{\mathtt{0}^n\#\mathtt{0}^{2n}\#\mathtt{0}^{3n} \mid n \geq 0\}$. Let p be the pumping length given by the pumping lemma. Let $s = \mathtt{0}^p\#\mathtt{0}^{2p}\#\mathtt{0}^{3p}$. We show that $s = uvxyz$ cannot be pumped.

Neither v nor y can contain #, otherwise xv^2wy^2z contains more than two #s. Therefore, if we divide s into three segments by #'s: $\mathtt{0}^p, \mathtt{0}^{2p}$, and $\mathtt{0}^{3p}$, at least one of the segments is not contained within either v or y. Hence xv^2wy^2z is not in B because the $1:2:3$ length ratio of the segments is not maintained.

(c) Let $C = \{w\#t \mid w \text{ is a substring of } t, \text{ where } w, t \in \{\mathtt{a}, \mathtt{b}\}^*\}$. Let p be the pumping length given by the pumping lemma. Let $s = \mathtt{a}^p\mathtt{b}^p\#\mathtt{a}^p\mathtt{b}^p$. We show that the string $s = uvxyz$ cannot be pumped.

Neither v nor y can contain #, otherwise uv^0xy^0z does not contain # and therefore is not in C. If both v and y are nonempty and occur on the left-hand side of the #, the string uv^2xy^2z cannot be in C because it is longer on the left-hand side of the #. Similarly, if both strings occur on the right-hand side of the #, the string uv^0xy^0z cannot be in C because it is again longer on the left-hand side of the #. If only one of v and y is nonempty (both cannot be nonempty), treat them as if both occurred on the same side of the # as above.

The only remaining case is where both v and y are nonempty and straddle the #. But then v consists of b's and y consists of a's because of the third pumping lemma condition $|vxy| \leq p$. Hence, uv^2xy^2z contains more b's on the left-hand side of the #, so it cannot be a member of C.

2.38 Let A be the language $\{\mathtt{0}^k\mathtt{1}^k \mid k \geq 0\}$ and let B be the language $\{\mathtt{a}^k\mathtt{b}^{3k} \mid k \geq 0\}$. The perfect shuffle of A and B is the language $C = \{(\mathtt{0a})^k(\mathtt{0b})^k(\mathtt{1b})^{2k} \mid k \geq 0\}$. Languages A and B are easily seen to be CFLs, but C is not a CFL, as follows. If C were a CFL, let p be the pumping length given by the pumping lemma, and let s be the string $(\mathtt{0a})^p(\mathtt{0b})^p(\mathtt{1b})^{2p}$. Because s is longer than p and $s \in C$, we can divide $s = uvxyz$ satisfying the pumping lemma's three conditions. Strings in C contain twice as many 1s as a's. In order for uv^2xy^2z to have that property, the string vxy must contain both 1s and a's. But that is impossible, because they are separated by $2p$ symbols yet the third condition says that $|vxy| \leq p$. Hence C is not context free.

Now M scans the list of edges. For each edge, M tests whether the two underlined nodes n_1 and n_2 are the ones appearing in that edge. If they are, M dots n_1, removes the underlines, and goes on from the beginning of stage 2. If they aren't, M checks the next edge on the list. If there are no more edges, $\{n_1, n_2\}$ is not an edge of G. Then M moves the underline on n_2 to the next dotted node and now calls this node n_2. It repeats the steps in this paragraph to check, as before, whether the new pair $\{n_1, n_2\}$ is an edge. If there are no more dotted nodes, n_1 is not attached to any dotted nodes. Then M sets the underlines so that n_1 is the next undotted node and n_2 is the first dotted node and repeats the steps in this paragraph. If there are no more undotted nodes, M has not been able to find any new nodes to dot, so it moves on to stage 4.

For stage 4, M scans the list of nodes to determine whether all are dotted. If they are, it enters the accept state; otherwise it enters the reject state. This completes the description of TM M.

EXERCISES

3.1 This exercise concerns TM M_2 whose description and state diagram appear in Example 3.7. In each of the parts, give the sequence of configurations that M_2 enters when started on the indicated input string.

- **a.** 0.
- [A]**b.** 00.
- **c.** 000.
- **d.** 000000.

3.2 This exercise concerns TM M_1 whose description and state diagram appear in Example 3.9. In each of the parts, give the sequence of configurations that M_1 enters when started on the indicated input string.

- [A]**a.** 11.
- **b.** 1#1.
- **c.** 1##1.
- **d.** 10#11.
- **e.** 10#10.

[A]**3.3** Modify the proof of Theorem 3.16 to obtain Corollary 3.19, showing that a language is decidable iff some nondeterministic Turing machine decides it. (You may assume the following theorem about trees. If every node in a tree has finitely many children and every branch of the tree has finitely many nodes, the tree itself has finitely many nodes.)

3.4 Give a formal definition of an enumerator. Consider it to be a type of two-tape Turing machine that uses its second tape as the printer. Include a definition of the enumerated language.

[A]**3.5** Examine the formal definition of a Turing machine to answer the following questions, and explain your reasoning.

a. Can a Turing machine ever write the blank symbol ␣ on its tape?
b. Can the tape alphabet Γ be the same as the input alphabet Σ?
c. Can a Turing machine's head *ever* be in the same location in two successive steps?
d. Can a Turing machine contain just a single state?

3.6 In Theorem 3.21 we showed that a language is Turing-recognizable iff some enumerator enumerates it. Why didn't we use the following simpler algorithm for the forward direction of the proof? As before, $s_1, s_2, \ldots$ is a list of all strings in Σ^*.

E = "Ignore the input.
1. Repeat the following for $i = 1, 2, 3, \ldots$
2. Run M on s_i.
3. If it accepts, print out s_i."

3.7 Explain why the following is not a description of a legitimate Turing machine.

M_{bad} = "The input is a polynomial p over variables $x_1, \ldots, x_k$.
1. Try all possible settings of $x_1, \ldots, x_k$ to integer values.
2. Evaluate p on all of these settings.
3. If any of these settings evaluates to 0, *accept*; otherwise, *reject*."

3.8 Give implementation-level descriptions of Turing machines that decide the following languages over the alphabet {0,1}.

[A]**a.** $\{w \mid w$ contains an equal number of 0s and 1s$\}$
b. $\{w \mid w$ contains twice as many 0s as 1s$\}$
c. $\{w \mid w$ does not contain twice as many 0s as 1s$\}$

PROBLEMS

3.9 Let a k-PDA be a pushdown automaton that has k stacks. Thus a 0-PDA is an NFA and a 1-PDA is a conventional PDA. You already know that 1-PDAs are more powerful (recognize a larger class of languages) than 0-PDAs.

a. Show that 2-PDAs are more powerful than 1-PDAs.
b. Show that 3-PDAs are not more powerful than 2-PDAs. (Hint: Simulate a Turing machine tape with two stacks.)

[A]**3.10** Say that a ***write-once Turing machine*** is a single-tape TM that can alter each tape square at most once (including the input portion of the tape). Show that this variant Turing machine model is equivalent to the ordinary Turing machine model. (Hint: As a first step consider the case whereby the Turing machine may alter each tape square at most twice. Use lots of tape.)

3.11 A ***Turing machine with doubly infinite tape*** is similar to an ordinary Turing machine, but its tape is infinite to the left as well as to the right. The tape is initially filled with blanks except for the portion that contains the input. Computation is defined as usual except that the head never encounters an end to the tape as it moves leftward. Show that this type of Turing machine recognizes the class of Turing-recognizable languages.

3.12 A ***Turing machine with left reset*** is similar to an ordinary Turing machine, but the transition function has the form

$$\delta\colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{R}, \mathrm{RESET}\}.$$

If $\delta(q, a) = (r, b, \mathrm{RESET})$, when the machine is in state q reading an a, the machine's head jumps to the left-hand end of the tape after it writes b on the tape and enters state r. Note that these machines do not have the usual ability to move the head one symbol left. Show that Turing machines with left reset recognize the class of Turing-recognizable languages.

3.13 A ***Turing machine with stay put instead of left*** is similar to an ordinary Turing machine, but the transition function has the form

$$\delta\colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{R}, \mathrm{S}\}.$$

At each point the machine can move its head right or let it stay in the same position. Show that this Turing machine variant is *not* equivalent to the usual version. What class of languages do these machines recognize?

3.14 A ***queue automaton*** is like a push-down automaton except that the stack is replaced by a queue. A ***queue*** is a tape allowing symbols to be written only on the left-hand end and read only at the right-hand end. Each write operation (we'll call it a *push*) adds a symbol to the left-hand end of the queue and each read operation (we'll call it a *pull*) reads and removes a symbol at the right-hand end. As with a PDA, the input is placed on a separate read-only input tape, and the head on the input tape can move only from left to right. The input tape contains a cell with a blank symbol following the input, so that the end of the input can be detected. A queue automaton accepts its input by entering a special accept state at any time. Show that a language can be recognized by a deterministic queue automaton iff the language is Turing-recognizable.

3.15 Show that the collection of decidable languages is closed under the operation of

[A]**a.** union.
b. concatenation.
c. star.
d. complementation.
e. intersection.

3.16 Show that the collection of Turing-recognizable languages is closed under the operation of

[A]**a.** union.
b. concatenation.
c. star.
d. intersection.

***3.17** Let $B = \{\langle M_1\rangle, \langle M_2\rangle, \ldots\}$ be a Turing-recognizable language consisting of TM descriptions. Show that there is a decidable language C consisting of TM descriptions such that every machine described in B has an equivalent machine in C and vice versa.

*3.18 Show that a language is decidable iff some enumerator enumerates the language in lexicographic order.

*3.19 Show that every infinite Turing-recognizable language has an infinite decidable subset.

*3.20 Show that single-tape TMs that cannot write on the portion of the tape containing the input string recognize only regular languages.

3.21 Let $c_1x^n + c_2x^{n-1} + \cdots + c_nx + c_{n+1}$ be a polynomial with a root at $x = x_0$. Let $c_{\max}$ be the largest absolute value of a c_i. Show that

$$|x_0| < (n+1)\frac{c_{\max}}{|c_1|}.$$

A3.22 Let A be the language containing only the single string s, where

$$s = \begin{cases} 0 & \text{if life never will be found on Mars.} \\ 1 & \text{if life will be found on Mars someday.} \end{cases}$$

Is A decidable? Why or why not? For the purposes of this problem, assume that the question of whether life will be found on Mars has an unambiguous YES or NO answer.

SELECTED SOLUTIONS

3.1 **(b)** $q_1\texttt{00}, \sqcup q_2\texttt{0}, \sqcup\texttt{x}q_3\sqcup, \sqcup q_5\texttt{x}\sqcup, q_5\sqcup\texttt{x}\sqcup, \sqcup q_2\texttt{x}\sqcup, \sqcup\texttt{x}q_2\sqcup, \sqcup\texttt{x}\sqcup q_{\text{accept}}$

3.2 **(a)** $q_1\texttt{11}, \texttt{x}q_3\texttt{1}, \texttt{x1}q_3\sqcup, \texttt{x1}\sqcup q_{\text{reject}}$.

3.3 We prove both directions of the "iff." First, if a language L is decidable, it can be decided by a deterministic Turing machine, and that is automatically a nondeterministic Turing machine.

Second, if a language L is decided by a nondeterministic TM N, we construct a deterministic TM D_2 that decides L. Machine D_2 runs the same algorithm that appears in the TM D described in the proof of Theorem 3.16, with an additional Stage 5: *Reject* if all branches of the nondeterminism of N are exhausted.

We argue that D_2 is a decider for L. If N accepts its input, D_2 will eventually find an accepting branch and accept, too. If N rejects its input, all of its branches halt and reject because it is a decider. Hence each of the branches has finitely many nodes, where each node represents one step of N's computation along that branch. Therefore N's entire computation tree on this input is finite, by virtue of the theorem about trees given in the statement of the exercise. Consequently D will halt and reject when this entire tree has been explored.

3.5 **(a)** Yes. The tape alphabet Γ contains $\sqcup$. A Turing machine can write any characters in Γ on its tape.

(b) No. Σ never contains $\sqcup$, but Γ always contains $\sqcup$. So they cannot be equal.

(c) Yes. If the Turing machine attempts to move its head off the left-hand end of the tape, it remains on the same tape cell.

(d) No. Any Turing machine must contain two distinct states q_{accept} and q_{reject}. So, a Turing machine contains at least two states.

3.8 (a) "On input string w:
1. Scan the tape and mark the first 0 which has not been marked. If no unmarked 0 is found, go to stage 4. Otherwise, move the head back to the front of the tape.
2. Scan the tape and mark the first 1 which has not been marked. If no unmarked 1 is found, *reject*.
3. Move the head back to the front of the tape and go to stage 1.
4. Move the head back to the front of the tape. Scan the tape to see if any unmarked 1s remain. If none are found, *accept*; otherwise, *reject*."

3.10 We first simulate an ordinary Turing machine by a write-twice Turing machine. The write-twice machine simulates a single step of the original machine by copying the entire tape over to a fresh portion of the tape to the right-hand side of the currently used portion. The copying procedure operates character by character, marking a character as it is copied. This procedure alters each tape square twice, once to write the character for the first time and again to mark that it has been copied. The position of the original Turing machine's tape head is marked on the tape. When copying the cells at, or adjacent to, the marked position, the tape contents is updated according to the rules of the original Turing machine.

To carry out the simulation with a write-once machine, operate as before, except that each cell of the previous tape is now represented by two cells. The first of these contains the original machine's tape symbol and the second is for the mark used in the copying procedure. The input is not presented to the machine in the format with two cells per symbol, so the very first time the tape is copied, the copying marks are put directly over the input symbols.

3.15 (a) For any two decidable languages L_1 and L_2, let M_1 and M_2 be the TMs that decide them. We construct a TM M' that decides the union of L_1 and L_2:

"On input w:
1. Run M_1 on w. If it accepts, *accept*.
2. Run M_2 on w. If it accepts, *accept*. Otherwise, *reject*."

M' accepts w if either M_1 or M_2 accepts it. If both reject, M' rejects.

3.16 (a) For any two Turing-recognizable languages L_1 and L_2, let M_1 and M_2 be the TMs that recognize them. We construct a TM M' that recognizes the union of L_1 and L_2:

"On input w:
1. Run M_1 and M_2 alternatively on w step by step. If either accept, *accept*. If both halt and reject, *reject*."

If either M_1 and M_2 accept w, M' accepts w because the accepting TM arrives to its accepting state after a finite number of steps. Note that if both M_1 and M_2 reject and either of them does so by looping, then M' will loop.

3.22 The language A is one of the two languages, $\{0\}$ or $\{1\}$. In either case the language is finite, and hence decidable. If you aren't able to determine which of these two languages is A, you won't be able to describe the decider for A, but you can give two Turing machines, one of which is A's decider.

machine M is a decider for A.

M = "On input w:

1. Run both M_1 and M_2 on input w in parallel.
2. If M_1 accepts, *accept*; if M_2 accepts, *reject*."

Running the two machines in parallel means that M has two tapes, one for simulating M_1 and the other for simulating M_2. In this case M takes turns simulating one step of each machine, which continues until one of them accepts.

Now we show that M decides A. Every string w is either in A or $\overline{A}$. Therefore either M_1 or M_2 must accept w. Because M halts whenever M_1 or M_2 accepts, M always halts and so it is a decider. Furthermore, it accepts all strings in A and rejects all strings not in A. So M is a decider for A, and thus A is decidable.

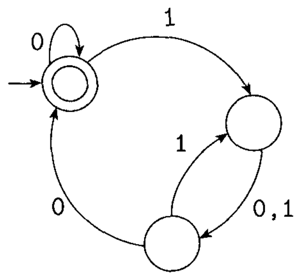
COROLLARY 4.23

$\overline{A_{\mathsf{TM}}}$ is not Turing-recognizable.

PROOF We know that A_{TM} is Turing-recognizable. If $\overline{A_{\mathsf{TM}}}$ also were Turing-recognizable, A_{TM} would be decidable. Theorem 4.11 tells us that A_{TM} is not decidable, so $\overline{A_{\mathsf{TM}}}$ must not be Turing-recognizable.

EXERCISES

[A]**4.1** Answer all parts for the following DFA M and give reasons for your answers.



a. Is $\langle M, 0100\rangle \in A_{\mathsf{DFA}}$?

b. Is $\langle M, 011\rangle \in A_{\mathsf{DFA}}$?

c. Is $\langle M\rangle \in A_{\mathsf{DFA}}$?

d. Is $\langle M, 0100\rangle \in A_{\mathsf{REX}}$?

e. Is $\langle M\rangle \in E_{\mathsf{DFA}}$?

f. Is $\langle M, M\rangle \in EQ_{\mathsf{DFA}}$?

4.2 Consider the problem of determining whether a DFA and a regular expression are equivalent. Express this problem as a language and show that it is decidable.

4.3 Let $ALL_{\text{DFA}} = \{\langle A\rangle \mid A$ is a DFA and $L(A) = \Sigma^*\}$. Show that ALL_{DFA} is decidable.

4.4 Let $A\varepsilon_{\text{CFG}} = \{\langle G\rangle \mid G$ is a CFG that generates $\varepsilon\}$. Show that $A\varepsilon_{\text{CFG}}$ is decidable.

4.5 Let X be the set $\{1,2,3,4,5\}$ and Y be the set $\{6,7,8,9,10\}$. We describe the functions $f\colon X\longrightarrow Y$ and $g\colon X\longrightarrow Y$ in the following tables. Answer each part and give a reason for each negative answer.

n	$f(n)$
1	6
2	7
3	6
4	7
5	6

n	$g(n)$
1	10
2	9
3	8
4	7
5	6

[A]**a.** Is f one-to-one?
b. Is f onto?
c. Is f a correspondence?
[A]**d.** Is g one-to-one?
e. Is g onto?
f. Is g a correspondence?

4.6 Let $\mathcal{B}$ be the set of all infinite sequences over $\{0,1\}$. Show that $\mathcal{B}$ is uncountable, using a proof by diagonalization.

4.7 Let $T = \{(i,j,k) \mid i,j,k \in \mathcal{N}\}$. Show that T is countable.

4.8 Review the way that we define sets to be the same size in Definition 4.12 (page 175). Show that "is the same size" is an equivalence relation.

PROBLEMS

[A]**4.9** Let $INFINITE_{\text{DFA}} = \{\langle A\rangle \mid A$ is a DFA and $L(A)$ is an infinite language$\}$. Show that $INFINITE_{\text{DFA}}$ is decidable.

4.10 Let $INFINITE_{\text{PDA}} = \{\langle M\rangle \mid M$ is a PDA and $L(M)$ is an infinite language$\}$. Show that $INFINITE_{\text{PDA}}$ is decidable.

[A]**4.11** Let $A = \{\langle M\rangle \mid M$ is a DFA which doesn't accept any string containing an odd number of 1s$\}$. Show that A is decidable.

4.12 Let $A = \{\langle R,S\rangle \mid R$ and S are regular expressions and $L(R) \subseteq L(S)\}$. Show that A is decidable.

[A]**4.13** Let $\Sigma = \{0,1\}$. Show that the problem of determining whether a CFG generates some string in 1^* is decidable. In other words, show that

$$\{\langle G\rangle \mid G \text{ is a CFG over } \{0,1\} \text{ and } 1^* \cap L(G) \neq \emptyset\}$$

is a decidable language.

*__4.14__ Show that the problem of determining whether a CFG generates all strings in 1^* is decidable. In other words, show that $\{\langle G\rangle \mid G$ is a CFG over $\{0,1\}$ and $1^* \subseteq L(G)\}$ is a decidable language.

4.15 Let $A = \{\langle R\rangle |\ R$ is a regular expression describing a language containing at least one string w that has 111 as a substring (i.e., $w = x111y$ for some x and y)$\}$. Show that A is decidable.

4.16 Prove that EQ_{DFA} is decidable by testing the two DFAs on all strings up to a certain size. Calculate a size that works.

***4.17** Let C be a language. Prove that C is Turing-recognizable iff a decidable language D exists such that $C = \{x |\ \exists y\ (\langle x, y\rangle \in D)\}$.

4.18 Let A and B be two disjoint languages. Say that language C ***separates*** A and B if $A \subseteq C$ and $B \subseteq \overline{C}$. Show that any two disjoint co-Turing-recognizable languages are separable by some decidable language.

4.19 Let $S = \{\langle M\rangle |\ M$ is a DFA that accepts $w^{\mathcal{R}}$ whenever it accepts $w\}$. Show that S is decidable.

4.20 A language is ***prefix-free*** if no member is a proper prefix of another member. Let $\mathit{PREFIX\text{-}FREE}_{\text{REX}} = \{R |\ R$ is a regular expression where $L(R)$ is prefix-free$\}$. Show that $\mathit{PREFIX\text{-}FREE}_{\text{REX}}$ is decidable. Why does a similar approach fail to show that $\mathit{PREFIX\text{-}FREE}_{\text{CFG}}$ is decidable?

A*4.21 Say that an NFA is ***ambiguous*** if it accepts some string along two different computation branches. Let $\mathit{AMBIG}_{\text{NFA}} = \{\langle N\rangle |\ N$ is an ambiguous NFA$\}$. Show that $\mathit{AMBIG}_{\text{NFA}}$ is decidable. (Suggestion: One elegant way to solve this problem is to construct a suitable DFA and then run E_{DFA} on it.)

4.22 A ***useless state*** in a pushdown automaton is never entered on any input string. Consider the problem of determining whether a pushdown automaton has any useless states. Formulate this problem as a language and show that it is decidable.

A*4.23 Let $\mathit{BAL}_{\text{DFA}} = \{\langle M\rangle |\ M$ is a DFA that accepts some string containing an equal number of 0s and 1s$\}$. Show that $\mathit{BAL}_{\text{DFA}}$ is decidable. (Hint: Theorems about CFLs are helpful here.)

***4.24** Let $\mathit{PAL}_{\text{DFA}} = \{\langle M\rangle |\ M$ is a DFA that accepts some palindrome$\}$. Show that $\mathit{PAL}_{\text{DFA}}$ is decidable. (Hint: Theorems about CFLs are helpful here.)

***4.25** Let $E = \{\langle M\rangle |\ M$ is a DFA that accepts some string with more 1s than 0s$\}$. Show that E is decidable. (Hint: Theorems about CFLs are helpful here.)

4.26 Let $C = \{\langle G, x\rangle |\ G$ is a CFG that generates some string w, where x is a substring of $w\}$. Show that C is decidable. (Suggestion: An elegant solution to this problem uses the decider for E_{CFG}.)

4.27 Let $C_{\text{CFG}} = \{\langle G, k\rangle |\ L(G)$ contains exactly k strings where $k \geq 0$ or $k = \infty\}$. Show that C_{CFG} is decidable.

4.28 Let A be a Turing-recognizable language consisting of descriptions of Turing machines, $\{\langle M_1\rangle, \langle M_2\rangle, \ldots\}$, where every M_i is a decider. Prove that some decidable language D is not decided by any decider M_i whose description appears in A. (Hint: You may find it helpful to consider an enumerator for A.)

SELECTED SOLUTIONS

4.1 **(a)** Yes. The DFA M accepts 0100.

(b) No. M doesn't accept 011.

(c) No. This input has only a single component and thus is not of the correct form.

(d) No. The first component is not a regular expression and so the input is not of the correct form.

(e) No. M's language isn't empty.

(f) Yes. M accepts the same language as itself.

4.5 **(a)** No, f is not one-to-one because $f(1) = f(3)$.

(d) Yes, g is one-to-one.

4.9 The following TM I decides $INFINITE_{\text{DFA}}$.

I = "On input $\langle A \rangle$ where A is a DFA:

1. Let k be the number of states of A.
2. Construct a DFA D that accepts all strings of length k or more.
3. Construct a DFA M such that $L(M) = L(A) \cap L(D)$.
4. Test $L(M) = \emptyset$, using the E_{DFA} decider T from Theorem 4.4.
5. If T accepts, *reject*; if T rejects, *accept*."

This algorithm works because a DFA which accepts infinitely many strings must accept arbitrarily long strings. Therefore this algorithm accepts such DFAs. Conversely, if the algorithm accepts a DFA, the DFA accepts some string of length k or more, where k is the number of states of the DFA. This string may be pumped in the manner of the pumping lemma for regular languages to obtain infinitely many accepted strings.

4.11 The following TM decides A.

"On input $\langle M \rangle$:

1. Construct a DFA O that accepts every string containing an odd number of 1s.
2. Construct DFA B such that $L(B) = L(M) \cap L(O)$.
3. Test whether $L(B) = \emptyset$, using the E_{DFA} decider T from Theorem 4.4.
4. If T accepts, *accept*; if T rejects, *reject*."

4.13 You showed in Problem 2.18 that, if C is a context-free language and R is a regular language, then $C \cap R$ is context free. Therefore $1^* \cap L(G)$ is context free. The following TM decides A.

"On input $\langle G \rangle$:

1. Construct CFG H such that $L(H) = 1^* \cap L(G)$.
2. Test whether $L(H) = \emptyset$, using the E_{CFG} decider R from Theorem 4.8.
3. If R accepts, *reject*; if R rejects, *accept*."

4.21 The following procedure decides $AMBIG_{\mathsf{NFA}}$. Given an NFA N, we design a DFA D that simulates N and accepts a string iff it is accepted by N along two different computational branches. Then we use a decider for E_{DFA} to determine whether D accepts any strings.

Our strategy for constructing D is similar to the NFA to DFA conversion in the proof of Theorem 1.39. We simulate N by keeping a pebble on each active state. We begin by putting a red pebble on the start state and on each state reachable from the start along ε transitions. We move, add, and remove pebbles in accordance with N's transitions, preserving the color of the pebbles. Whenever two or more pebbles are moved to the same state, we replace its pebbles with a blue pebble. After reading the input, we accept if a blue pebble is on an accept states of N.

The DFA D has a state corresponding to each possible position of pebbles. For each state of N, three possibilities occur: it can contain a red pebble, a blue pebble, or no pebble. Thus, if N has n states, D will have 3^n states. Its start state, accept states, and transition function are defined to carry out the simulation.

4.23 The language of all strings with an equal number of 0s and 1s is a context-free language, generated by the grammar $S \rightarrow 1S0S \mid 0S1S \mid \varepsilon$. Let P be the PDA that recognizes this language. Build a TM M for BAL_{DFA}, which operates as follows. On input $\langle B \rangle$, where B is a DFA, use B and P to construct a new PDA R that recognizes the intersection of the languages of B and P. Then test whether R's language is empty. If its language is empty, *reject*; otherwise, *accept*.

EXERCISES

5.1 Show that EQ_{CFG} is undecidable.

5.2 Show that EQ_{CFG} is co-Turing-recognizable.

5.3 Find a match in the following instance of the Post Correspondence Problem.

$$\left\{ \left[\frac{\mathtt{ab}}{\mathtt{abab}}\right], \left[\frac{\mathtt{b}}{\mathtt{a}}\right], \left[\frac{\mathtt{aba}}{\mathtt{b}}\right], \left[\frac{\mathtt{aa}}{\mathtt{a}}\right] \right\}$$

5.4 If $A \leq_{\mathrm{m}} B$ and B is a regular language, does that imply that A is a regular language? Why or why not?

[A]5.5 Show that A_{TM} is not mapping reducible to E_{TM}. In other words, show that no computable function reduces A_{TM} to E_{TM}. (Hint: Use a proof by contradiction, and facts you already know about A_{TM} and E_{TM}.)

[A]5.6 Show that $\leq_{\mathrm{m}}$ is a transitive relation.

[A]5.7 Show that if A is Turing-recognizable and $A \leq_{\mathrm{m}} \overline{A}$, then A is decidable.

[A]5.8 In the proof of Theorem 5.15 we modified the Turing machine M so that it never tries to move its head off the left-hand end of the tape. Suppose that we did not make this modification to M. Modify the PCP construction to handle this case.

PROBLEMS

5.9 Let $T = \{\langle M \rangle \mid M \text{ is a TM that accepts } w^{\mathcal{R}} \text{ whenever it accepts } w\}$. Show that T is undecidable.

[A]5.10 Consider the problem of determining whether a two-tape Turing machine ever writes a nonblank symbol on its second tape when it is run on input w. Formulate this problem as a language, and show that it is undecidable.

[A]5.11 Consider the problem of determining whether a two-tape Turing machine ever writes a nonblank symbol on its second tape during the course of its computation on any input string. Formulate this problem as a language, and show that it is undecidable.

5.12 Consider the problem of determining whether a single-tape Turing machine ever writes a blank symbol over a nonblank symbol during the course of its computation on any input string. Formulate this problem as a language, and show that it is undecidable.

5.13 A ***useless state*** in a Turing machine is one that is never entered on any input string. Consider the problem of determining whether a Turing machine has any useless states. Formulate this problem as a language and show that it is undecidable.

5.14 Consider the problem of determining whether a Turing machine M on an input w ever attempts to move its head left when its head is on the left-most tape cell. Formulate this problem as a language and show that it is undecidable.

5.15 Consider the problem of determining whether a Turing machine M on an input w ever attempts to move its head left at any point during its computation on w. Formulate this problem as a language and show that it *is* decidable.

5.16 Let $\Gamma = \{0, 1, \sqcup\}$ be the tape alphabet for all TMs in this problem. Define the ***busy beaver function*** $BB\colon \mathcal{N} \longrightarrow \mathcal{N}$ as follows. For each value of k, consider all k-state TMs that halt when started with a blank tape. Let $BB(k)$ be the maximum number of 1s that remain on the tape among all of these machines. Show that BB is not a computable function.

5.17 Show that the Post Correspondence Problem is decidable over the unary alphabet $\Sigma = \{1\}$.

5.18 Show that the Post Correspondence Problem is undecidable over the binary alphabet $\Sigma = \{0,1\}$.

5.19 In the *silly Post Correspondence Problem*, *SPCP*, in each pair the top string has the same length as the bottom string. Show that the *SPCP* is decidable.

5.20 Prove that there exists an undecidable subset of $\{1\}^*$.

5.21 Let $AMBIG_{\mathsf{CFG}} = \{\langle G\rangle \mid G \text{ is an ambiguous CFG}\}$. Show that $AMBIG_{\mathsf{CFG}}$ is undecidable. (Hint: Use a reduction from *PCP*. Given an instance

$$P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots, \left[\frac{t_k}{b_k}\right] \right\},$$

of the Post Correspondence Problem, construct a CFG G with the rules

$$\begin{aligned} S &\rightarrow T \mid B \\ T &\rightarrow t_1 T \mathtt{a}_1 \mid \cdots \mid t_k T \mathtt{a}_k \mid t_1 \mathtt{a}_1 \mid \cdots \mid t_k \mathtt{a}_k \\ B &\rightarrow b_1 B \mathtt{a}_1 \mid \cdots \mid b_k B \mathtt{a}_k \mid b_1 \mathtt{a}_1 \mid \cdots \mid b_k \mathtt{a}_k \ , \end{aligned}$$

where $\mathtt{a}_1, \ldots, \mathtt{a}_k$ are new terminal symbols. Prove that this reduction works.)

5.22 Show that A is Turing-recognizable iff $A \le_{\mathrm{m}} A_{\mathsf{TM}}$.

5.23 Show that A is decidable iff $A \le_{\mathrm{m}} 0^*1^*$.

5.24 Let $J = \{w \mid \text{either } w = 0x \text{ for some } x \in A_{\mathsf{TM}}, \text{ or } w = 1y \text{ for some } y \in \overline{A_{\mathsf{TM}}}\ \}$. Show that neither J nor $\overline{J}$ is Turing-recognizable.

5.25 Give an example of an undecidable language B, where $B \le_{\mathrm{m}} \overline{B}$.

5.26 Define a ***two-headed finite automaton*** (2DFA) to be a deterministic finite automaton that has two read-only, bidirectional heads that start at the left-hand end of the input tape and can be independently controlled to move in either direction. The tape of a 2DFA is finite and is just large enough to contain the input plus two additional blank tape cells, one on the left-hand end and one on the right-hand end, that serve as delimiters. A 2DFA accepts its input by entering a special accept state. For example, a 2DFA can recognize the language $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \ge 0\}$.

a. Let $A_{\mathsf{2DFA}} = \{\langle M, x\rangle \mid M \text{ is a 2DFA and } M \text{ accepts } x\}$. Show that A_{2DFA} is decidable.

b. Let $E_{\mathsf{2DFA}} = \{\langle M\rangle \mid M \text{ is a 2DFA and } L(M) = \emptyset\}$. Show that E_{2DFA} is not decidable.

5.27 A ***two-dimensional finite automaton*** (2DIM-DFA) is defined as follows. The input is an $m \times n$ rectangle, for any $m, n \geq 2$. The squares along the boundary of the rectangle contain the symbol # and the internal squares contain symbols over the input alphabet Σ. The transition function is a mapping $Q \times \Sigma \rightarrow Q \times \{\text{L}, \text{R}, \text{U}, \text{D}\}$ to indicate the next state and the new head position (Left, Right, Up, Down). The machine accepts when it enters one of the designated accept states. It rejects if it tries to move off the input rectangle or if it never halts. Two such machines are equivalent if they accept the same rectangles. Consider the problem of determining whether two of these machines are equivalent. Formulate this problem as a language, and show that it is undecidable.

A*5.28 **Rice's theorem.** Let P be any nontrivial property of the language of a Turing machine. Prove that the problem of determining whether a given Turing machine's language has property P is undecidable.

In more formal terms, let P be a language consisting of Turing machine descriptions where P fulfills two conditions. First, P is nontrivial—it contains some, but not all, TM descriptions. Second, P is a property of the TM's language—whenever $L(M_1) = L(M_2)$, we have $\langle M_1 \rangle \in P$ iff $\langle M_2 \rangle \in P$. Here, M_1 and M_2 are any TMs. Prove that P is an undecidable language.

5.29 Show that both conditions in Problem 5.28 are necessary for proving that P is undecidable.

5.30 Use Rice's theorem, which appears in Problem 5.28, to prove the undecidability of each of the following languages.

- A**a.** $INFINITE_{\text{TM}} = \{\langle M \rangle |\ M \text{ is a TM and } L(M) \text{ is an infinite language}\}$.
- **b.** $\{\langle M \rangle |\ M \text{ is a TM and } 1011 \in L(M)\}$.
- **c.** $ALL_{\text{TM}} = \{\langle M \rangle |\ M \text{ is a TM and } L(M) = \Sigma^*\}$.

5.31 Let

$$f(x) = \begin{cases} 3x + 1 & \text{for odd } x \\ x/2 & \text{for even } x \end{cases}$$

for any natural number x. If you start with an integer x and iterate f, you obtain a sequence, $x, f(x), f(f(x)), \ldots$ Stop if you ever hit 1. For example, if $x = 17$, you get the sequence 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1. Extensive computer tests have shown that every starting point between 1 and a large positive integer gives a sequence that ends in 1. But, the question of whether all positive starting points end up at 1 is unsolved; it is called the $3x + 1$ problem.

Suppose that A_{TM} were decidable by a TM H. Use H to describe a TM that is guaranteed to state the answer to the $3x + 1$ problem.

5.32 Prove that the following two languages are undecidable.

- **a.** $OVERLAP_{\text{CFG}} = \{\langle G, H \rangle |\ G \text{ and } H \text{ are CFGs where } L(G) \cap L(H) \neq \emptyset\}$. (Hint: Adapt the hint in Problem 5.21.)
- **b.** $PREFIX\text{-}FREE_{\text{CFG}} = \{G |\ G \text{ is a CFG where } L(G) \text{ is prefix-free}\}$.

5.33 Let $S = \{\langle M \rangle |\ M \text{ is a TM and } L(M) = \{\langle M \rangle\}\ \}$. Show that neither S nor $\overline{S}$ is Turing-recognizable.

5.34 Consider the problem of determining whether a PDA accepts some string of the form $\{ww |\ w \in \{0,1\}^*\}$. Use the computation history method to show that this problem is undecidable.

5.35 Let $X = \{\langle M, w\rangle |\ M$ is a single-tape TM that never modifies the portion of the tape that contains the input $w\}$. Is X decidable? Prove your answer.

SELECTED SOLUTIONS

5.5 Suppose for a contradiction that $A_{\mathsf{TM}} \leq_{\mathrm{m}} E_{\mathsf{TM}}$ via reduction f. It follows from the definition of mapping reducibility that $\overline{A_{\mathsf{TM}}} \leq_{\mathrm{m}} \overline{E_{\mathsf{TM}}}$ via the same reduction function f. However $\overline{E_{\mathsf{TM}}}$ is Turing-recognizable and $\overline{A_{\mathsf{TM}}}$ is not Turing-recognizable, contradicting Theorem 5.28.

5.6 Suppose $A \leq_{\mathrm{m}} B$ and $B \leq_{\mathrm{m}} C$. Then there are computable functions f and g such that $x \in A \iff f(x) \in B$ and $y \in B \iff g(y) \in C$. Consider the composition function $h(x) = g(f(x))$. We can build a TM that computes h as follows: First, simulate a TM for f (such a TM exists because we assumed that f is computable) on input x and call the output y. Then simulate a TM for g on y. The output is $h(x) = g(f(x))$. Therefore h is a computable function. Moreover, $x \in A \iff h(x) \in C$. Hence $A \leq_{\mathrm{m}} C$ via the reduction function h.

5.7 Suppose that $A \leq_{\mathrm{m}} \overline{A}$. Then $\overline{A} \leq_{\mathrm{m}} A$ via the same mapping reduction. Because A is Turing-recognizable, Theorem 5.28 implies that $\overline{A}$ is Turing-recognizable, and then Theorem 4.22 implies that A is decidable.

5.8 You need to handle the case where the head is at the leftmost tape cell and attempts to move left. To do so add dominos

$$\left[\frac{\#qa}{\#rb}\right]$$

for every $q, r \in Q$ and $a, b \in \Gamma$, where $\delta(q, a) = (r, b, \mathrm{L})$.

5.10 Let $B = \{\langle M, w\rangle |\ M$ is a two-tape TM which writes a nonblank symbol on its second tape when it is run on $w\}$. Show that A_{TM} reduces to B. Assume for the sake of contradiction that TM R decides B. Then construct TM S that uses R to decide A_{TM}.

$S =$ "On input $\langle M, w\rangle$:
1. Use M to construct the following two-tape TM T.
   $T =$ "On input x:
   1. Simulate M on x using the first tape.
   2. If the simulation shows that M accepts, write a non-blank symbol on the second tape."
2. Run R on $\langle T, w\rangle$ to determine whether T on input w writes a nonblank symbol on its second tape.
3. If R accepts, M accepts w, therefore *accept*. Otherwise *reject*."

5.11 Let $C = \{\langle M \rangle |\ M$ is a two-tape TM which writes a nonblank symbol on its second tape when it is run on some input$\}$. Show that A_{TM} reduces to C. Assume for the sake of contradiction that TM R decides C. Construct TM S that uses R to decide A_{TM}.

$S =$ "On input $\langle M, w \rangle$:

1. Use M and w to construct the following two-tape TM T_w.
   $T_w =$ "On any input:
   1. Simulate M on w using the first tape.
   2. If the simulation shows that M accepts, write a non-blank symbol on the second tape."
2. Run R on $\langle T_w \rangle$ to determine whether T_w ever writes a nonblank symbol on its second tape.
3. If R accepts, M accepts w, therefore *accept*. Otherwise *reject*."

5.28 Assume for the sake of contradiction that P is a decidable language satisfying the properties and let R_P be a TM that decides P. We show how to decide A_{TM} using R_P by constructing TM S. First let $T_\emptyset$ be a TM that always rejects, so $L(T_\emptyset) = \emptyset$. You may assume that $\langle T_\emptyset \rangle \notin P$ without loss of generality, because you could proceed with $\overline{P}$ instead of P if $\langle T_\emptyset \rangle \in P$. Because P is not trivial, there exists a TM T with $\langle T \rangle \in P$. Design S to decide A_{TM} using R_P's ability to distinguish between $T_\emptyset$ and T.

$S =$ "On input $\langle M, w \rangle$:

1. Use M and w to construct the following TM M_w.
   $M_w =$ "On input x:
   1. Simulate M on w. If it halts and rejects, *reject*. If it accepts, proceed to stage 2.
   2. Simulate T on x. If it accepts, *accept*."
2. Use TM R_P to determine whether $\langle M_w \rangle \in P$. If YES, *accept*. If NO, *reject*."

TM M_w simulates T if M accepts w. Hence $L(M_w)$ equals $L(T)$ if M accepts w and $\emptyset$ otherwise. Therefore $\langle M, w \rangle \in P$ iff M accepts w.

5.30 **(a)** $\mathit{INFINITE}_{\mathsf{TM}}$ is a language of TM descriptions. It satisfies the two conditions of Rice's theorem. First, it is nontrivial because some TMs have infinite languages and others do not. Second, it depends only on the language. If two TMs recognize the same language, either both have descriptions in $\mathit{INFINITE}_{\mathsf{TM}}$ or neither do. Consequently, Rice's theorem implies that $\mathit{INFINITE}_{\mathsf{TM}}$ is undecidable.

EXERCISES

6.1 Give an example in the spirit of the recursion theorem of a program in a real programming language (or a reasonable approximation thereof) that prints itself out.

6.2 Show that any infinite subset of MIN_{TM} is not Turing-recognizable.

A6.3 Show that if $A \leq_{\text{T}} B$ and $B \leq_{\text{T}} C$ then $A \leq_{\text{T}} C$.

6.4 Let $A_{\text{TM}}{}' = \{\langle M, w\rangle \mid M$ is an oracle TM and $M^{A_{\text{TM}}}$ accepts $w\}$. Show that $A_{\text{TM}}{}'$ is undecidable relative to A_{TM}.

A6.5 Is the statement $\exists x\, \forall y\, \big[\, x{+}y{=}y \,\big]$ a member of $\text{Th}(\mathcal{N}, +)$? Why or why not? What about the statement $\exists x\, \forall y\, \big[\, x{+}y{=}x \,\big]$?

PROBLEMS

6.6 Describe two different Turing machines, M and N, that, when started on any input, M outputs $\langle N\rangle$ and N outputs $\langle M\rangle$.

6.7 In the fixed-point version of the recursion theorem (Theorem 6.8) let the transformation t be a function that interchanges the states q_{accept} and q_{reject} in Turing machine descriptions. Give an example of a fixed point for t.

***6.8** Show that $EQ_{\text{TM}} \not\leq_{\text{m}} \overline{EQ_{\text{TM}}}$.

A6.9 Use the recursion theorem to give an alternative proof of Rice's theorem in Problem 5.28.

A6.10 Give a model of the sentence

$$\begin{aligned}\phi_{\text{eq}} = \quad & \forall x\, \big[\, R_1(x,x) \,\big] \\ & \land \forall x{,}y \big[\, R_1(x,y) \leftrightarrow R_1(y,x) \,\big] \\ & \land \forall x{,}y{,}z \big[\, (R_1(x,y) \land R_1(y,z)) \rightarrow R_1(x,z) \,\big].\end{aligned}$$

***6.11** Let ϕ_{eq} be defined as in Problem 6.10. Give a model of the sentence

$$\begin{aligned}\phi_{\text{lt}} = \quad & \phi_{\text{eq}} \\ & \land \forall x{,}y\, \big[\, R_1(x,y) \rightarrow \neg R_2(x,y) \,\big] \\ & \land \forall x{,}y\, \big[\, \neg R_1(x,y) \rightarrow (R_2(x,y) \oplus R_2(y,x)) \,\big] \\ & \land \forall x{,}y{,}z\, \big[\, (R_2(x,y) \land R_2(y,z)) \rightarrow R_2(x,z) \,\big] \\ & \land \forall x\, \exists y\, \big[\, R_2(x,y) \,\big].\end{aligned}$$

A6.12 Let $(\mathcal{N}, <)$ be the model with universe $\mathcal{N}$ and the "less than" relation. Show that $\text{Th}(\mathcal{N}, <)$ is decidable.

6.13 For each $m > 1$ let $\mathcal{Z}_m = \{0, 1, 2, \ldots, m-1\}$ and let $\mathcal{F}_m = (\mathcal{Z}_m, +, \times)$ be the model whose universe is $\mathcal{Z}_m$ and that has relations corresponding to the $+$ and $\times$ relations computed modulo m. Show that for each m the theory $\text{Th}(\mathcal{F}_m)$ is decidable.

6.14 Show that for any two languages A and B a language J exists, where $A \leq_T J$ and $B \leq_T J$.

6.15 Show that for any language A, a language B exists, where $A \leq_T B$ and $B \not\leq_T A$.

***6.16** Prove that there exist two languages A and B that are Turing-incomparable—that is, where $A \not\leq_T B$ and $B \not\leq_T A$.

***6.17** Let A and B be two disjoint languages. Say that language C *separates* A and B if $A \subseteq C$ and $B \subseteq \overline{C}$. Describe two disjoint Turing-recognizable languages that aren't separable by any decidable language.

6.18 In Corollary 4.18 we showed that the set of all languages is uncountable. Use this result to prove that languages exist that are not recognizable by an oracle Turing machine with oracle for A_{TM}.

6.19 Recall the Post correspondence problem that we defined in Section 5.2 and its associated language PCP. Show that PCP is decidable relative to A_{TM}.

6.20 Show how to compute the descriptive complexity of strings $\text{K}(x)$ with an oracle for A_{TM}.

6.21 Use the result of Problem 6.20 to give a function f that is computable with an oracle for A_{TM}, where for each n, $f(n)$ is an incompressible string of length n.

6.22 Show that the function $\text{K}(x)$ is not a computable function.

6.23 Show that the set of incompressible strings is undecidable.

6.24 Show that the set of incompressible strings contains no infinite subset that is Turing-recognizable.

***6.25** Show that for any c, some strings x and y exist, where $\text{K}(xy) > \text{K}(x) + \text{K}(y) + c$.

SELECTED SOLUTIONS

6.3 Say that M_1^B decides A and M_2^C decides B. Use an oracle TM M_3, where M_3^C decides A. Machine M_3 simulates M_1. Every time M_1 queries its oracle about some string x, machine M_3 tests whether $x \in B$ and provides the answer to M_1. Because machine M_3 doesn't have an oracle for B and cannot perform that test directly, it simulates M_2 on input x to obtain that information. Machine M_3 can obtain the answer to M_2's queries directly because these two machines use the same oracle, C.

6.5 The statement $\exists x \forall y \left[x{+}y{=}y \right]$ is a member of $\text{Th}(\mathcal{N}, +)$ because that statement is true for the standard interpretation of $+$ over the universe $\mathcal{N}$. Recall that we use $\mathcal{N} = \{0, 1, 2, \ldots\}$ in this chapter and so we may use $x = 0$. The statement $\exists x \forall y \left[x{+}y{=}x \right]$ is not a member of $\text{Th}(\mathcal{N}, +)$ because that statement isn't true in this model. For any value of x, setting $y = 1$ causes $x{+}y{=}x$ to fail.

6.9 Assume for the sake of contradiction that some TM X decides a property P, and P satisfies the conditions of Rice's theorem. One of these conditions says that TMs A and B exist where $\langle A\rangle \in P$ and $\langle B\rangle \notin P$. Use A and B to construct TM R:

$R =$ "On input w:
1. Obtain own description $\langle R\rangle$ using the recursion theorem.
2. Run X on $\langle R\rangle$.
3. If X accepts $\langle R\rangle$, simulate B on w.
   If X rejects $\langle R\rangle$, simulate A on w."

If $\langle R\rangle \in P$, then X accepts $\langle R\rangle$ and $L(R) = L(B)$. But $\langle B\rangle \notin P$, contradicting $\langle R\rangle \in P$, because P agrees on TMs that have the same language. We arrive at a similar contradiction if $\langle R\rangle \notin P$. Therefore our original assumption is false. Every property satisfying the conditions of Rice's theorem is undecidable.

6.10 The statement ϕ_{eq} gives the three conditions of an equivalence relation. A model (A, R_1), where A is any universe and R_1 is any equivalence relation over A, is a model of ϕ_{eq}. For example, let A be the integers $\mathcal{Z}$ and let $R_1 = \{(i,i) |\ i \in \mathcal{Z}\}$.

6.12 Reduce $\text{Th}(\mathcal{N}, <)$ to $\text{Th}(\mathcal{N}, +)$, which we've already shown to be decidable. To do so, show how to convert a sentence ϕ_1 over the language of $\text{Th}(\mathcal{N}, <)$, to a sentence ϕ_2 over the language of $\text{Th}(\mathcal{N}, +)$ while preserving truth or falsity in the respective models. Replace every occurrence of $i < j$ in ϕ_1 by the formula $\exists k\big[\,(i{+}k{=}j) \land (k{+}k{\neq}k)\,\big]$ in ϕ_2, where k is a different new variable each time.

Sentence ϕ_2 is equivalent to ϕ_1 because "i is less than j" means that we can add a nonzero value to i and obtain j. Putting ϕ_2 into prenex-normal form, as required by the algorithm for deciding $\text{Th}(\mathcal{N}, +)$, requires a bit of additional work. The new existential quantifiers are brought to the front of the sentence. To do so, these quantifiers must pass through Boolean operations that appear in the sentence. Quantifiers can be brought through the operations of $\land$ and $\lor$ without change. Passing through $\neg$ changes $\exists$ to $\forall$ and vice-versa. Thus $\neg\exists k\,\psi$ becomes the equivalent expression $\forall k\,\neg\psi$, and $\neg\forall k\,\psi$ becomes $\exists k\,\neg\psi$.

y_i or z_i for each i, but not both.

Now we make the satisfying assignment. If the subset contains y_i, we assign x_i TRUE; otherwise, we assign it FALSE. This assignment must satisfy ϕ because in each of the final k columns the sum is always 3. In column c_j, at most 2 can come from g_j and h_j, so at least 1 in this column must come from some y_i or z_i in the subset. If it is y_i, then x_i appears in c_j and is assigned TRUE, so c_j is satisfied. If it is z_i, then $\overline{x_i}$ appears in c_j and x_i is assigned FALSE, so c_j is satisfied. Therefore ϕ is satisfied.

Finally, we must be sure that the reduction can be carried out in polynomial time. The table has a size of roughly $(k + l)^2$, and each entry can be easily calculated for any ϕ. So the total time is $O(n^2)$ easy stages.

EXERCISES

7.1 Answer each part TRUE or FALSE.

- **a.** $2n = O(n)$.
- **b.** $n^2 = O(n)$.
- ^A**c.** $n^2 = O(n \log^2 n)$.
- ^A**d.** $n \log n = O(n^2)$.
- **e.** $3^n = 2^{O(n)}$.
- **f.** $2^{2^n} = O(2^{2^n})$.

7.2 Answer each part TRUE or FALSE.

- **a.** $n = o(2n)$.
- **b.** $2n = o(n^2)$.
- ^A**c.** $2^n = o(3^n)$.
- ^A**d.** $1 = o(n)$.
- **e.** $n = o(\log n)$.
- **f.** $1 = o(1/n)$.

7.3 Which of the following pairs of numbers are relatively prime? Show the calculations that led to your conclusions.

- **a.** 1274 and 10505
- **b.** 7289 and 8029

7.4 Fill out the table described in the polynomial time algorithm for context-free language recognition from Theorem 7.16 for string $w =$ baba and CFG G:

$$S \rightarrow RT$$
$$R \rightarrow TR \mid \mathtt{a}$$
$$T \rightarrow TR \mid \mathtt{b}$$

7.5 Is the following formula satisfiable?

$$(x \vee y) \wedge (x \vee \overline{y}) \wedge (\overline{x} \vee y) \wedge (\overline{x} \vee \overline{y})$$

7.6 Show that P is closed under union, concatenation, and complement.

7.7 Show that NP is closed under union and concatenation.

7.8 Let $CONNECTED = \{\langle G\rangle |\ G$ is a connected undirected graph$\}$. Analyze the algorithm given on page 157 to show that this language is in P.

7.9 A ***triangle*** in an undirected graph is a 3-clique. Show that $TRIANGLE \in$ P, where $TRIANGLE = \{\langle G\rangle |\ G$ contains a triangle$\}$.

7.10 Show that ALL_{DFA} is in P.

7.11 Call graphs G and H ***isomorphic*** if the nodes of G may be reordered so that it is identical to H. Let $ISO = \{\langle G, H\rangle |\ G$ and H are isomorphic graphs$\}$. Show that $ISO \in$ NP.

PROBLEMS

7.12 Let

$$MODEXP = \{\langle a, b, c, p\rangle |\ a, b, c, \text{ and } p \text{ are binary integers such that } a^b \equiv c \pmod{p}\}.$$

Show that $MODEXP \in$ P. (Note that the most obvious algorithm doesn't run in polynomial time. Hint: Try it first where b is a power of 2.)

7.13 A ***permutation*** on the set $\{1, \ldots, k\}$ is a one-to-one, onto function on this set. When p is a permutation, p^t means the composition of p with itself t times. Let

$$PERM\text{-}POWER = \{\langle p, q, t\rangle |\ p = q^t \text{ where } p \text{ and } q \text{ are permutations on } \{1, \ldots, k\} \text{ and } t \text{ is a binary integer}\}.$$

Show that $PERM\text{-}POWER \in$ P. (Note that the most obvious algorithm doesn't run within polynomial time. Hint: First try it where t is a power of 2).

7.14 Show that P is closed under the star operation. (Hint: Use dynamic programming. On input $y = y_1 \cdots y_n$ for $y_i \in \Sigma$, build a table indicating for each $i \leq j$ whether the substring $y_i \cdots y_j \in A^*$ for any $A \in$ P.)

A7.15 Show that NP is closed under the star operation.

7.16 Let $UNARY\text{-}SSUM$ be the subset sum problem in which all numbers are represented in unary. Why does the NP-completeness proof for $SUBSET\text{-}SUM$ fail to show $UNARY\text{-}SSUM$ is NP-complete? Show that $UNARY\text{-}SSUM \in$ P.

7.17 Show that, if P = NP, then every language $A \in$ P, except $A = \emptyset$ and $A = \Sigma^*$, is NP-complete.

***7.18** Show that $PRIMES = \{m | m$ is a prime number in binary$\} \in$ NP. (Hint: For $p > 1$ the multiplicative group $Z_p^* = \{x |\ x$ is relatively prime to p and $1 \leq x < p\}$ is both cyclic and of order $p - 1$ iff p is prime. You may use this fact without justifying it. The stronger statement $PRIMES \in$ P is now known to be true, but it is more difficult to prove.)

7.19 We generally believe that $PATH$ is not NP-complete. Explain the reason behind this belief. Show that proving $PATH$ is not NP-complete would prove P $\neq$ NP.

7.20 Let G represent an undirected graph. Also let

$$SPATH = \{\langle G, a, b, k\rangle \mid G \text{ contains a simple path of length at most } k \text{ from } a \text{ to } b\},$$

and

$$LPATH = \{\langle G, a, b, k\rangle \mid G \text{ contains a simple path of length at least } k \text{ from } a \text{ to } b\}.$$

a. Show that $SPATH \in \mathrm{P}$.

b. Show that $LPATH$ is NP-complete. You may assume the NP-completeness of *UHAMPATH*, the Hamiltonian path problem for undirected graphs.

7.21 Let $DOUBLE\text{-}SAT = \{\langle\phi\rangle \mid \phi$ has at least two satisfying assignments$\}$. Show that *DOUBLE-SAT* is NP-complete.

[A]**7.22** Let $HALF\text{-}CLIQUE = \{\langle G\rangle \mid G$ is an undirected graph having a complete subgraph with at least $m/2$ nodes, where m is the number of nodes in $G\}$. Show that *HALF-CLIQUE* is NP-complete.

7.23 Let $CNF_k = \{\langle\phi\rangle \mid \phi$ is a satisfiable cnf-formula where each variable appears in at most k places$\}$.

a. Show that $CNF_2 \in \mathrm{P}$.

b. Show that CNF_3 is NP-complete.

7.24 Let ϕ be a 3cnf-formula. An ***≠-assignment*** to the variables of ϕ is one where each clause contains two literals with unequal truth values. In other words, an ≠-assignment satisfies ϕ without assigning three true literals in any clause.

a. Show that the negation of any ≠-assignment to ϕ is also an ≠-assignment.

b. Let $\neq SAT$ be the collection of 3cnf-formulas that have an ≠-assignment. Show that we obtain a polynomial time reduction from *3SAT* to $\neq SAT$ by replacing each clause c_i

$$(y_1 \vee y_2 \vee y_3)$$

with the two clauses

$$(y_1 \vee y_2 \vee z_i) \quad \text{and} \quad (\overline{z_i} \vee y_3 \vee b),$$

where z_i is a new variable for each clause c_i and b is a single additional new variable.

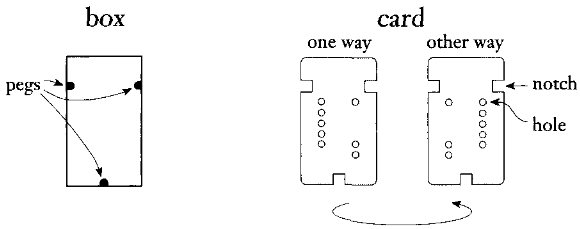
c. Conclude that $\neq SAT$ is NP-complete.

7.25 A ***cut*** in an undirected graph is a separation of the vertices V into two disjoint subsets S and T. The size of a cut is the number of edges that have one endpoint in S and the other in T. Let

$$MAX\text{-}CUT = \{\langle G, k\rangle \mid G \text{ has a cut of size } k \text{ or more}\}.$$

Show that *MAX-CUT* is NP-complete. You may assume the result of Problem 7.24. (Hint: Show that $\neq SAT \leq_{\mathrm{P}} MAX\text{-}CUT$. The variable gadget for variable x is a collection of $3c$ nodes labeled with x and another $3c$ nodes labeled with $\overline{x}$, where c is the number of clauses. All nodes labeled x are connected with all nodes labeled $\overline{x}$. The clause gadget is a triangle of three edges connecting three nodes labeled with the literals appearing in the clause. Do not use the same node in more than one clause gadget. Prove that this reduction works.)

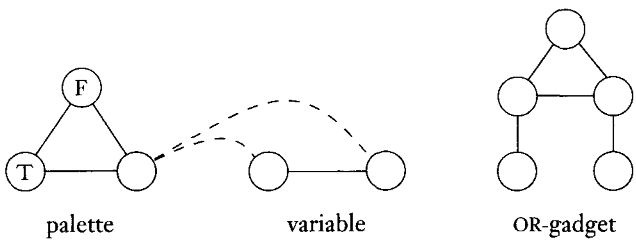
7.26 You are given a box and a collection of cards as indicated in the following figure. Because of the pegs in the box and the notches in the cards, each card will fit in the box in either of two ways. Each card contains two columns of holes, some of which may not be punched out. The puzzle is solved by placing all the cards in the box so as to completely cover the bottom of the box, (i.e., every hole position is blocked by at least one card that has no hole there.) Let *PUZZLE* $= \{\langle c_1, \ldots, c_k\rangle |$ each c_i represents a card and this collection of cards has a solution$\}$. Show that *PUZZLE* is NP-complete.



7.27 A ***coloring*** of a graph is an assignment of colors to its nodes so that no two adjacent nodes are assigned the same color. Let

$$3COLOR = \{\langle G\rangle |\text{ the nodes of } G \text{ can be colored with three colors such that no two nodes joined by an edge have the same color}\}.$$

Show that *3COLOR* is NP-complete. (Hint: Use the following three subgraphs.)



7.28 Let *SET-SPLITTING* $= \{\langle S, C\rangle |$ S is a finite set and $C = \{C_1, \ldots, C_k\}$ is a collection of subsets of S, for some $k > 0$, such that elements of S can be colored *red* or *blue* so that no C_i has all its elements colored with the same color.$\}$ Show that *SET-SPLITTING* is NP-complete.

7.29 Consider the following scheduling problem. You are given a list of final exams $F_1, \ldots, F_k$ to be scheduled, and a list of students $S_1, \ldots, S_l$. Each student is taking some specified subset of these exams. You must schedule these exams into slots so that no student is required to take two exams in the same slot. The problem is to determine if such a schedule exists that uses only h slots. Formulate this problem as a language and show that this language is NP-complete.

7.30 This problem is inspired by the single-player game *Minesweeper*, generalized to an arbitrary graph. Let G be an undirected graph, where each node either contains a single, hidden *mine* or is empty. The player chooses nodes, one by one. If the player chooses a node containing a mine, the player loses. If the player chooses an empty node, the player learns the number of neighboring nodes containing mines. (A neighboring node is one connected to the chosen node by an edge.). The player wins if and when all empty nodes have been so chosen.

In the *mine consistency problem* you are given a graph G, along with numbers labeling some of G's nodes. You must determine whether a placement of mines on the remaining nodes is possible, so that any node v that is labeled m has exactly m neighboring nodes containing mines. Formulate this problem as a language and show that it is NP-complete.

^{A}**7.31** In the following solitaire game, you are given an $m \times m$ board. On each of its n^2 positions lies either a blue stone, a red stone, or nothing at all. You play by removing stones from the board so that each column contains only stones of a single color and each row contains at least one stone. You win if you achieve this objective. Winning may or may not be possible, depending upon the initial configuration. Let $SOLITAIRE = \{\langle G \rangle |\ G \text{ is a winnable game configuration}\}$. Prove that *SOLITAIRE* is NP-complete.

7.32 Let $U = \{\langle M, x, \#^t \rangle |\ \text{TM } M \text{ accepts input } x \text{ within } t \text{ steps on at least one branch}\}$. Show that U is NP-complete.

7.33 Recall, in our discussion of the Church–Turing thesis, that we introduced the language $D = \{\langle p \rangle |\ p \text{ is a polynomial in several variables having an integral root}\}$. We stated, but didn't prove, that D is undecidable. In this problem you are to prove a different property of D—namely, that D is NP-hard. A problem is ***NP-hard*** if all problems in NP are polynomial time reducible to it, even though it may not be in NP itself. So, you must show that all problems in NP are polynomial time reducible to D.

7.34 A subset of the nodes of a graph G is a ***dominating set*** if every other node of G is adjacent to some node in the subset. Let

$$DOMINATING\text{-}SET = \{\langle G, k \rangle |\ G \text{ has a dominating set with } k \text{ nodes}\}.$$

Show that it is NP-complete by giving a reduction from *VERTEX-COVER*.

***7.35** Show that the following problem is NP-complete. You are given a set of states $Q = \{q_0, q_1, \ldots, q_l\}$ and a collection of pairs $\{(s_1, r_1), \ldots, (s_k, r_k)\}$ where the s_i are distinct strings over $\Sigma = \{0, 1\}$, and the r_i are (not necessarily distinct) members of Q. Determine whether a DFA $M = (Q, \Sigma, \delta, q_0, F)$ exists where $\delta(q_0, s_i) = r_i$ for each i. Here, $\delta(q, s)$ is the state that M enters after reading s, starting at state q. (Note that F is irrelevant here).

***7.36** Show that if P = NP, a polynomial time algorithm exists that produces a satisfying assignment when given a satisfiable Boolean formula. (Note: The algorithm you are asked to provide computes a function, but NP contains languages, not functions. The P = NP assumption implies that *SAT* is in P, so testing satisfiability is solvable in polynomial time. But the assumption doesn't say how this test is done, and the test may not reveal satisfying assignments. You must show that you can find them anyway. Hint: Use the satisfiability tester repeatedly to find the assignment bit-by-bit.)

*7.37 Show that if P = NP, you can factor integers in polynomial time. (See the note in Problem 7.36.)

A*7.38 Show that if P = NP, a polynomial time algorithm exists that takes an undirected graph as input and finds a largest clique contained in that graph. (See the note in Problem 7.36.)

7.39 In the proof of the Cook–Levin theorem, a window is a 2×3 rectangle of cells. Show why the proof would have failed if we had used 2×2 windows instead.

*7.40 Consider the algorithm *MINIMIZE*, which takes a DFA M as input and outputs DFA M'.

MINIMIZE = "On input $\langle M \rangle$, where $M = (Q, \Sigma, \delta, q_0, A)$ is a DFA:
1. Remove all states of M that are unreachable from the start state.
2. Construct the following undirected graph G whose nodes are the states of M.
3. Place an edge in G connecting every accept state with every nonaccept state. Add additional edges as follows.
4. Repeat until no new edges are added to G:
5. For every pair of distinct states q and r of M and every $a \in \Sigma$:
6. Add the edge (q, r) to G if $(\delta(q, a), \delta(r, a))$ is an edge of G.
7. For each state q, let $[q]$ be the collection of states $[q] = \{r \in Q |$ no edge joins q and r in $G\}$.
8. Form a new DFA $M' = (Q', \Sigma, \delta', q_0', A')$ where
$Q' = \{[q] | \, q \in Q\}$, (if $[q] = [r]$, only one of them is in Q'),
$\delta'([q], a) = [\delta(q, a)]$, for every $q \in Q$ and $a \in \Sigma$,
$q_0' = [q_0]$, and
$A' = \{[q] | \, q \in A\}$.
9. Output $\langle M' \rangle$."

a. Show that M and M' are equivalent.
b. Show that M' is minimal—that is, no DFA with fewer states recognizes the same language. You may use the result of Problem 1.52 without proof.
c. Show that *MINIMIZE* operates in polynomial time.

7.41 For a cnf-formula ϕ with m variables and c clauses, show that you can construct in polynomial time an NFA with $O(cm)$ states that accepts all nonsatisfying assignments, represented as Boolean strings of length m. Conclude that NFAs cannot be minimized in polynomial time unless P = NP.

*7.42 A ***2cnf-formula*** is an AND of clauses, where each clause is an OR of at most two literals. Let $2SAT = \{\langle \phi \rangle | \, \phi$ is a satisfiable 2cnf-formula$\}$. Show that $2SAT \in$ P.

7.43 Modify the algorithm for context-free language recognition in the proof of Theorem 7.16 to give a polynomial time algorithm that produces a parse tree for a string, given the string and a CFG, if that grammar generates the string.

7.44 Say that two Boolean formulas are ***equivalent*** if they have the same set of variables and are true on the same set of assignments to those variables (i.e., they describe the same Boolean function). A Boolean formula is ***minimal*** if no shorter Boolean formula is equivalent to it. Let *MIN-FORMULA* be the collection of minimal Boolean formulas. Show that, if P = NP, then *MIN-FORMULA* $\in$ P.

7.45 The ***difference hierarchy*** D_iP is defined recursively as

a. $D_1P = NP$ and

b. $D_iP = \{A \mid A = B \setminus C$ for B in NP and C in $D_{i-1}P\}$.
(Here $B \setminus C = B \cap \overline{C}$.)

For example, a language in D_2P is the difference of two NP languages. Sometimes D_2P is called DP (and may be written D^P). Let

$$Z = \{\langle G_1, k_1, G_2, k_2\rangle \mid G_1 \text{ has a } k_1\text{-clique and } G_2 \text{ doesn't have a } k_2\text{-clique}\}.$$

Show that Z is complete for DP. In other words, show that every language in DP is polynomial time reducible to Z.

***7.46** Let *MAX-CLIQUE* $= \{\langle G, k\rangle \mid$ the largest clique in G is of size exactly $k\}$. Use the result of Problem 7.45 to show that *MAX-CLIQUE* is DP-complete.

***7.47** Let $f\colon \mathcal{N} \longrightarrow \mathcal{N}$ be any function where $f(n) = o(n \log n)$. Show that TIME$(f(n))$ contains only the regular languages.

7.48** Call a regular expression ***star-free if it does not contain any star operations. Let $EQ_{\mathsf{SF-REX}} = \{\langle R, S\rangle \mid R$ and S are equivalent star-free regular expressions$\}$. Show that $EQ_{\mathsf{SF-REX}}$ is in coNP. Why does your argument fail for general regular expressions?

7.49** This problem investigates ***resolution, a method for proving the unsatisfiability of cnf-formulas. Let $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be a formula in cnf, where the C_i are its clauses. Let $\mathcal{C} = \{C_i \mid C_i$ is a clause of $\phi\}$. In a *resolution step* we take two clauses C_a and C_b in $\mathcal{C}$ which both have some variable x, occurring positively in one of the clauses and negatively in the other. Thus $C_a = (x \vee y_1 \vee y_2 \vee \cdots \vee y_k)$ and $C_b = (\overline{x} \vee z_1 \vee z_2 \vee \cdots \vee z_l)$, where the y_i and z_i are literals. We form the new clause $(y_1 \vee y_2 \vee \cdots \vee y_k \vee z_1 \vee z_2 \vee \cdots \vee z_l)$ and remove repeated literals. Add this new clause to $\mathcal{C}$. Repeat the resolution steps until no additional clauses can be obtained. If the empty clause ( ) is in $\mathcal{C}$ then declare ϕ unsatisfiable.

Say that resolution is ***sound*** if it never declares satisfiable formulas to be unsatisfiable. Say that resolution is ***complete*** if all unsatisfiable formulas are declared to be unsatisfiable.

a. Show that resolution is sound and complete.

b. Use part (a) to show that *2SAT* $\in$ P.

SELECTED SOLUTIONS

7.1 **(c)** FALSE; **(d)** TRUE.

7.2 **(c)** TRUE; **(d)** TRUE.

7.15 Let $A \in$ NP. Construct NTM M to decide A in nondeterministic polynomial time.

M = "On input w:

1. Nondeterministically divide w into pieces $w = x_1x_2 \cdots x_k$.
2. For each x_i, nondeterministically guess the certificates that show $x_i \in A$.
3. Verify all certificates if possible, then *accept*. Otherwise if verification fails, *reject*."

7.22 We give a polynomial time mapping reduction from *CLIQUE* to *HALF-CLIQUE*. The input to the reduction is a pair $\langle G, k\rangle$ and the reduction produces the graph $\langle H\rangle$ as output where H is as follows. If G has m nodes and $k = m/2$ then $H = G$. If $k < m/2$, then H is the graph obtained from G by adding j nodes, each connected to every one of the original nodes and to each other, where $j = m - 2k$. Thus H has $m + j = 2m - 2k$ nodes. Observe that G has a k-clique iff H has a clique of size $k+j = m-k$ and so $\langle G, k\rangle \in$ *CLIQUE* iff $\langle H\rangle \in$ *HALF-CLIQUE*. If $k > 2m$, then H is the graph obtained by adding j nodes to G without any additional edges, where $j = 2k - m$. Thus H has $m + j = 2k$ nodes, and so G has a k-clique iff H has a clique of size k. Therefore $\langle G, k_k\rangle \in$ *CLIQUE* iff $\langle H\rangle \in$ *HALF-CLIQUE*. We also need to show *HALF-CLIQUE* $\in$ NP. The certificate is simply the clique.

7.31 First, *SOLITAIRE* $\in$ NP because we can verify that a solution works, in polynomial time. Second, we show that *3SAT* $\leq_{\text{P}}$ *SOLITAIRE*. Given ϕ with m variables $x_1, \ldots, x_m$ and k clauses $c_1, \ldots, c_k$, construct the following $k \times m$ game G. We assume that ϕ has no clauses that contain both x_i and $\overline{x_i}$ because such clauses may be removed without affecting satisfiability.

If x_i is in clause c_j put a blue stone in row c_j, column x_i. If $\overline{x_i}$ is in clause c_j put a red stone in row c_j, column x_i. We can make the board square by repeating a row or adding a blank column as necessary without affecting solvability. We show that ϕ is satisfiable iff G has a solution.

($\rightarrow$) Take a satisfying assignment. If x_i is true (false), remove the red (blue) stones from the corresponding column. So, stones corresponding to true literals remain. Because every clause has a true literal, every row has a stone.

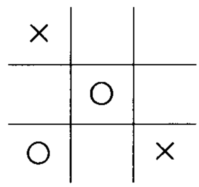
($\leftarrow$) Take a game solution. If the red (blue) stones were removed from a column, set the corresponding variable true (false). Every row has a stone remaining, so every clause has a true literal. Therefore ϕ is satisfied.

7.38 If you assume that P = NP, then *CLIQUE* $\in$ P, and you can test whether G contains a clique of size k in polynomial time, for any value of k. By testing whether G contains a clique of each size, from 1 to the number of nodes in G, you can determine the size t of a maximum clique in G in polynomial time. Once you know t, you can find a clique with t nodes as follows. For each node x of G, remove x and calculate the resulting maximum clique size. If the resulting size decreases, replace x and continue with the next node. If the resulting size is still t, keep x permanently removed and continue with the next node. When you have considered all nodes in this way, the remaining nodes are a t-clique.

EXERCISES

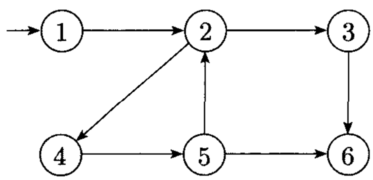
8.1 Show that for any function $f\colon \mathcal{N} \longrightarrow \mathcal{R}^+$, where $f(n) \geq n$, the space complexity class SPACE($f(n)$) is the same whether you define the class by using the single-tape TM model or the two tape read-only input TM model.

8.2 Consider the following position in the standard tic-tac-toe game.



Let's say that it is the X-player's turn to move next. Describe the winning strategy for this player. (Recall that a winning strategy isn't merely the best move to make in the current position. It also includes all the responses that this player must make in order to win, however the opponent moves.)

8.3 Consider the following generalized geography game wherein the start node is the one with the arrow pointing in from nowhere. Does Player I have a winning strategy? Does Player II? Give reasons for your answers.



8.4 Show that PSPACE is closed under the operations union, complementation, and star.

[A]**8.5** Show that NL is closed under the operations union, intersection, and star.

8.6 Show that any PSPACE-hard language is also NP-hard.

[A]**8.7** Show that $A_{\mathsf{DFA}} \in$ L.

PROBLEMS

8.8 Let $EQ_{\mathsf{REX}} = \{\langle R, S\rangle \mid R \text{ and } S \text{ are equivalent regular expressions}\}$. Show that $EQ_{\mathsf{REX}} \in$ PSPACE.

8.9 A *ladder* is a sequence of strings $s_1, s_2, \ldots, s_k$, wherein every string differs from the preceding one in exactly one character. For example the following is a ladder of English words, starting with "head" and ending with "free":

head, hear, near, fear, bear, beer, deer, deed, feed, feet, fret, free.

Let $LADDER_{\text{DFA}} = \{\langle M, s, t\rangle |\ M$ is a DFA and $L(M)$ contains a ladder of strings, starting with s and ending with $t\}$. Show that $LADDER_{\text{DFA}}$ is in PSPACE.

8.10 The Japanese game *go-moku* is played by two players, "X" and "O," on a 19×19 grid. Players take turns placing markers, and the first player to achieve 5 of his markers consecutively in a row, column, or diagonal, is the winner. Consider this game generalized to an $n \times n$ board. Let

$$GM = \{\langle B\rangle |\ B \text{ is a position in generalized go-moku, where player "X" has a winning strategy}\}.$$

By a *position* we mean a board with markers placed on it, such as may occur in the middle of a play of the game, together with an indication of which player moves next. Show that $GM \in$ PSPACE.

8.11 Show that, if every NP-hard language is also PSPACE-hard, then PSPACE = NP.

8.12 Show that *TQBF* restricted to formulas where the part following the quantifiers is in conjunctive normal form is still PSPACE-complete.

8.13 Define $A_{\text{LBA}} = \{\langle M, w\rangle |\ M$ is an LBA that accepts input $w\}$. Show that A_{LBA} is PSPACE-complete.

8.14 Consider the following two-person version of the language *PUZZLE* that was described in Problem 7.26. Each player starts with an ordered stack of puzzle cards. The players take turns placing the cards in order in the box and may choose which side faces up. Player I wins if, in the final stack, all hole positions are blocked, and Player II wins if some hole position remains unblocked. Show that the problem of determining which player has a winning strategy for a given starting configuration of the cards is PSPACE-complete.

***8.15** The cat-and-mouse game is played by two players, "Cat" and "Mouse," on an arbitrary undirected graph. At a given point each player occupies a node of the graph. The players take turns moving to a node adjacent to the one that they currently occupy. A special node of the graph is called "Hole." Cat wins if the two players ever occupy the same node. Mouse wins if it reaches the Hole before the preceding happens. The game is a draw if a situation repeats (i.e., the two players simultaneously occupy positions that they simultaneously occupied previously and it is the same player's turn to move).

$$HAPPY\text{-}CAT = \{\langle G, c, m, h\rangle |\ G, c, m, h, \text{ are respectively a graph, and positions of the Cat, Mouse, and Hole, such that Cat has a winning strategy if Cat moves first}\}.$$

Show that *HAPPY-CAT* is in P. (Hint: The solution is not complicated and doesn't depend on subtle details in the way the game is defined. Consider the entire game tree. It is exponentially big, but you can search it in polynomial time.)

8.16 Read the definition of *MIN-FORMULA* in Problem 7.44.

a. Show that *MIN-FORMULA* $\in$ PSPACE.

b. Explain why this argument fails to show that *MIN-FORMULA* $\in$ coNP: If $\phi \notin$ *MIN-FORMULA*, then ϕ has a smaller equivalent formula. An NTM can verify that $\phi \in \overline{\textit{MIN-FORMULA}}$ by guessing that formula.

8.17 Let A be the language of properly nested parentheses. For example, `(())` and `(()(()))()` are in A, but `)(` is not. Show that A is in L.

***8.18** Let B be the language of properly nested parentheses and brackets. For example, `([()()]()[])` is in B but `([)]` is not. Show that B is in L.

8.19** The game of ***Nim is played with a collection of piles of sticks. In one move a player may remove any nonzero number of sticks from a single pile. The players alternately take turns making moves. The player who removes the very last stick loses. Say that we have a game position in Nim with k piles containing $s_1, \ldots, s_k$ sticks. Call the position ***balanced*** if, when each of the numbers s_i is written in binary and the binary numbers are written as rows of a matrix aligned at the low order bits, each column of bits contains an even number of 1s. Prove the following two facts.

a. Starting in an unbalanced position, a single move exists that changes the position into a balanced one.

b. Starting in a balanced position, every single move changes the position into an unbalanced one.

Let $\textit{NIM} = \{\langle s_1, \ldots, s_k\rangle |$ each s_i is a binary number and Player I has a winning strategy in the Nim game starting at this position$\}$. Use the preceding facts about balanced positions to show that $\textit{NIM} \in$ L.

8.20 Let $\textit{MULT} = \{a\#b\#c|$ where a, b, c are binary natural numbers and $a \times b = c\}$. Show that $\textit{MULT} \in$ L.

8.21 For any positive integer x, let $x^{\mathcal{R}}$ be the integer whose binary representation is the reverse of the binary representation of x. (Assume no leading 0s in the binary representation of x.) Define the function $\mathcal{R}^+ : \mathcal{N} \longrightarrow \mathcal{N}$ where $\mathcal{R}^+(x) = x + x^{\mathcal{R}}$.

a. Let $A_2 = \{\langle x, y\rangle |\ \mathcal{R}^+(x) = y\}$. Show $A_2 \in$ L.

b. Let $A_3 = \{\langle x, y\rangle |\ \mathcal{R}^+(\mathcal{R}^+(x)) = y\}$. Show $A_3 \in$ L.

8.22 **a.** Let $\textit{ADD} = \{\langle x, y, z\rangle |\ x, y, z > 0$ are binary integers and $x + y = z\}$. Show that $\textit{ADD} \in$ L.

b. Let $\textit{PAL-ADD} = \{\langle x, y\rangle |\ x, y > 0$ are binary integers where $x + y$ is an integer whose binary representation is a palindrome$\}$. (Note that the binary representation of the sum is assumed not to have leading zeros. A palindrome is a string that equals its reverse). Show that $\textit{PAL-ADD} \in$ L.

***8.23** Define $\textit{UCYCLE} = \{\langle G\rangle |\ G$ is an undirected graph that contains a simple cycle$\}$. Show that $\textit{UCYCLE} \in$ L. (Note: G may be a graph that is not connected.)

***8.24** For each n, exhibit two regular expressions, R and S, of length $poly(n)$, where $L(R) \neq L(S)$, but where the first string on which they differ is exponentially long. In other words, $L(R)$ and $L(S)$ must be different, yet agree on all strings of length $2^{\epsilon n}$ for some constant $\epsilon > 0$.

8.25 An undirected graph is ***bipartite*** if its nodes may be divided into two sets so that all edges go from a node in one set to a node in the other set. Show that a graph is bipartite if and only if it doesn't contain a cycle that has an odd number of nodes. Let $BIPARTITE = \{\langle G \rangle | \ G \text{ is a bipartite graph}\}$. Show that $BIPARTITE \in$ NL.

8.26 Define *UPATH* to be the counterpart of *PATH* for undirected graphs. Show that $BIPARTITE \leq_{\text{L}} UPATH$. (Note: As this edition was going to press, O. Reingold [60] announced that $UPATH \in$ L. Consequently, $BIPARTITE \in$ L, but the algorithm is somewhat complicated.)

8.27 Recall that a directed graph is ***strongly connected*** if every two nodes are connected by a directed path in each direction. Let

$$STRONGLY\text{-}CONNECTED = \{\langle G \rangle | \ G \text{ is a strongly connected graph}\}.$$

Show that *STRONGLY-CONNECTED* is NL-complete.

8.28 Let $BOTH_{\text{NFA}} = \{\langle M_1, M_2 \rangle | \ M_1 \text{ and } M_2 \text{ are NFAs where } L(M_1) \cap L(M_2) \neq \emptyset\}$. Show that $BOTH_{\text{NFA}}$ is NL-complete.

8.29 Show that A_{NFA} is NL-complete.

8.30 Show that E_{DFA} is NL-complete.

***8.31** Show that *2SAT* is NL-complete.

***8.32** Give an example of an NL-complete context-free language.

A***8.33** Define $CYCLE = \{\langle G \rangle | \ G \text{ is a directed graph that contains a directed cycle}\}$. Show that *CYCLE* is NL-complete.

SELECTED SOLUTIONS

8.5 Let A_1 and A_2 be languages that are decided by NL-machines N_1 and N_2. Construct three Turing machines: $N_\cup$ deciding $A_1 \cup A_2$; $N_\circ$ deciding $A_1 \circ A_2$; and N_* deciding A_1^*. Each of these machines receives input w.

Machine $N_\cup$ nondeterministically branches to simulate N_1 or to simulate N_2. In either case, $N_\cup$ accepts if the simulated machine accepts.

Machine $N_\circ$ nondeterministically selects a position on the input to divide it into two substrings. Only a pointer to that position is stored on the work tape—insufficient space is available to store the substrings themselves. Then $N_\circ$ simulates N_1 on the first substring, branching nondeterministically to simulate N_1's nondeterminism. On any branch that reaches N_1's accept state, $N_\circ$ simulates N_2 on the second substring. On any branch that reaches N_2's accept state, $N_\circ$ accepts.

Machine N_* has a more complex algorithm, so we describe its stages.

N_* = "On input w:

1. Initialize two input position pointers p_1 and p_2 to 0, the position immediately preceding the first input symbol.
2. *Accept* if no input symbols occur after p_2.
3. Move p_2 forward to a nondeterministically selected input position.

4. Simulate N_1 on the substring of w from the position following p_1 to the position at p_2, branching nondeterministically to simulate N_1's nondeterminism.
5. If this branch of the simulation reaches N_1's accept state, copy p_2 to p_1 and go to stage 2."

8.7 Construct a TM M to decide A_{DFA}. When M receives input $\langle A, w\rangle$, a DFA and a string, M simulates A on w by keeping track of A's current state and its current head location, and updating them appropriately. The space required to carry out this simulation is $O(\log n)$ because M can record each of these values by storing a pointer into its input.

8.33 Reduce *PATH* to *CYCLE*. The idea behind the reduction is to modify the *PATH* problem instance $\langle G, s, t\rangle$ by adding an edge from t to s in G. If a path exists from s to t in G, a directed cycle will exist in the modified G. However, other cycles may exist in the modified G because they may already be present in G. To handle that problem, first change G so that it contains no cycles. A ***leveled directed graph*** is one where the nodes are divided into groups, $A_1, A_2, \ldots, A_k$, called *levels*, and only edges from one level to the next higher level are permitted. Observe that a leveled graph is acyclic. The *PATH* problem for leveled graphs is still NL-complete, as the following reduction from the unrestricted *PATH* problem shows. Given a graph G with two nodes s and t, and m nodes in total, produce the leveled graph G' whose levels are m copies of G's nodes. Draw an edge from node i at each level to node j in the next level if G contains an edge from i to j. Additionally, draw an edge from node i in each level to node i in the next level. Let s' be the node s in the first level and let t' be the node t in the last level. Graph G contains a path from s to t iff G' contains a path from s' to t'. If you modify G' by adding an edge from t' to s', you obtain a reduction from *PATH* to *CYCLE*. The reduction is computationally simple, and its implementation in logspace is routine. Furthermore, a straightforward procedure shows that *CYCLE* $\in$ NL. Hence *CYCLE* is NL-complete.

EXERCISES

[A]9.1 Prove that $\text{TIME}(2^n) = \text{TIME}(2^{n+1})$.

[A]9.2 Prove that $\text{TIME}(2^n) \subsetneq \text{TIME}(2^{2n})$.

[A]9.3 Prove that $\text{NTIME}(n) \subsetneq \text{PSPACE}$.

9.4 Show how the circuit depicted in Figure 9.26 computes on input 0110 by showing the values computed by all of the gates, as we did in Figure 9.24.

9.5 Give a circuit that computes the parity function on three input variables and show how it computes on input 011.

9.6 Prove that if $A \in \text{P}$ then $\text{P}^A = \text{P}$.

9.7 Give regular expressions with exponentiation that generate the following languages over the alphabet {0,1}.

- [A]**a.** All strings of length 500
- [A]**b.** All strings of length 500 or less
- [A]**c.** All strings of length 500 or more
- [A]**d.** All strings of length different than 500
- **e.** All strings that contain exactly 500 1s
- **f.** All strings that contain at least 500 1s
- **g.** All strings that contain at most 500 1s
- **h.** All strings of length 500 or more that contain a 0 in the 500th position
- **i.** All strings that contain two 0s that have at least 500 symbols between them

9.8 If R is a regular expression, let $R^{\{m,n\}}$ represent the expression

$$R^m \cup R^{m+1} \cup \cdots \cup R^n.$$

Show how to implement the $R^{\{m,n\}}$ operator, using the ordinary exponentiation operator, but without "$\cdots$".

9.9 Show that if $\text{NP} = \text{P}^{SAT}$, then $\text{NP} = \text{coNP}$.

9.10 Problem 8.13 showed that A_{LBA} is PSPACE-complete.

- **a.** Do we know whether $A_{\text{LBA}} \in \text{NL}$? Explain your answer.
- **b.** Do we know whether $A_{\text{LBA}} \in \text{P}$? Explain your answer.

9.11 Show that the language *MAX-CLIQUE* from Problem 7.46 is in P^{SAT}.

PROBLEMS

9.12 Describe the error in the following fallacious "proof" that P≠NP. Assume that P=NP and obtain a contradiction. If P=NP, then $SAT \in \text{P}$ and so for some k, $SAT \in \text{TIME}(n^k)$. Because every language in NP is polynomial time reducible to SAT, you have $\text{NP} \subseteq \text{TIME}(n^k)$. Therefore $\text{P} \subseteq \text{TIME}(n^k)$. But, by the time hierarchy theorem, $\text{TIME}(n^{k+1})$ contains a language that isn't in $\text{TIME}(n^k)$, which contradicts $\text{P} \subseteq \text{TIME}(n^k)$. Therefore $\text{P} \neq \text{NP}$.

9.13 Consider the function $pad\colon \Sigma^* \times \mathcal{N} \longrightarrow \Sigma^*\#^*$ that is defined as follows. Let $pad(s,l) = s\#^j$, where $j = \max(0, l - m)$ and m is the length of s. Thus $pad(s,l)$ simply adds enough copies of the new symbol # to the end of s so that the length of the result is at least l. For any language A and function $f\colon \mathcal{N} \longrightarrow \mathcal{N}$ define the language $pad(A, f(m))$ as

$$pad(A, f(m)) = \{pad(s, f(m)) |\text{ where } s \in A \text{ and } m \text{ is the length of } s\}.$$

Prove that, if $A \in \text{TIME}(n^6)$, then $pad(A, n^2) \in \text{TIME}(n^3)$.

9.14 Prove that, if NEXPTIME $\neq$ EXPTIME, then P $\neq$ NP. You may find the function *pad*, defined in Problem 9.13, to be helpful.

A9.15 Define *pad* as in Problem 9.13.

a. Prove that, for every A and natural number k, $A \in \text{P}$ iff $pad(A, n^k) \in \text{P}$.

b. Prove that P $\neq$ SPACE(n).

9.16 Prove that $TQBF \notin \text{SPACE}(n^{1/3})$

***9.17** Read the definition of a 2DFA (two-headed finite automaton) given in Problem 5.26. Prove that P contains a language that is not recognizable by a 2DFA.

9.18 Let $E_{\text{REG}\uparrow} = \{\langle R\rangle |\ R \text{ is a regular expression with exponentiation and } L(R) = \emptyset\}$. Show that $E_{\text{REG}\uparrow} \in \text{P}$.

9.19 Define the ***unique-sat*** problem to be

$$USAT = \{\langle\phi\rangle |\ \phi \text{ is a Boolean formula that has a single satisfying assignment}\}.$$

Show that $USAT \in \text{P}^{SAT}$.

9.20 Prove that an oracle C exists for which $\text{NP}^C \neq \text{coNP}^C$.

9.21 A ***k-query oracle Turing machine*** is an oracle Turing machine that is permitted to make at most k queries on each input. A k-oracle TM M with an oracle for A is written $M^{A,k}$ and $\text{P}^{A,k}$ is the collection of languages that are decidable by polynomial time k-oracle A TMs.

a. Show that $\text{NP} \cup \text{coNP} \subseteq \text{P}^{SAT,1}$.

b. Assume that NP $\neq$ coNP. Show that $P \cup \text{coNP} \subsetneq \text{P}^{SAT,1}$.

9.22 Suppose that A and B are two oracles. One of them is an oracle for $TQBF$, but you don't know which. Give an algorithm that has access to both A and B and that is guaranteed to solve $TQBF$ in polynomial time.

9.23 Define the function $parity_n$ as in Example 9.25. Show that $parity_n$ can be computed with $O(n)$ size circuits.

9.24 Recall that you may consider circuits that output strings over {0,1} by designating several output gates. Let $add_n\colon \{0,1\}^{2n} \longrightarrow \{0,1\}^{n+1}$ take the sum of two n bit binary integers and produce the $n+1$ bit result. Show that you can compute the add_n function with $O(n)$ size circuits.

9.25 Define the function $majority_n \colon \{0,1\}^n \longrightarrow \{0,1\}$ as

$$majority_n(x_1, \ldots, x_n) = \begin{cases} 0 & \sum x_i < n/2\,; \\ 1 & \sum x_i \geq n/2\,. \end{cases}$$

Thus the $majority_n$ function returns the majority vote of the inputs. Show that $majority_n$ can be computed with

a. $O(n^2)$ size circuits.

b. $O(n \log n)$ size circuits. (Hint: Recursively divide the number of inputs in half and use the result of Problem 9.24.)

***9.26** Define the problem $majority_n$ as in Problem 9.25. Show that it may be computed with $O(n)$ size circuits.

SELECTED SOLUTIONS

9.1 The time complexity classes are defined in terms of the big-O notation, so constant factors have no effect. The function 2^{n+1} is $O(2^n)$ and thus $A \in \text{TIME}(2^n)$ iff $A \in \text{TIME}(2^{n+1})$.

9.2 The containment $\text{TIME}(2^n) \subseteq \text{TIME}(2^{2n})$ holds because $2^n \leq 2^{2n}$. The containment is proper by virtue of the time hierarchy theorem. The function 2^{2n} is time constructible because a TM can write the number 1 followed by $2n$ 0s in $O(2^{2n})$ time. Hence the theorem guarantees that a language A exists that can be decided in $O(2^{2n})$ time but not in $O(2^{2n}/\log 2^{2n}) = O(2^{2n}/2n)$ time. Therefore $A \in \text{TIME}(2^{2n})$ but $A \notin \text{TIME}(2^n)$.

9.3 $\text{NTIME}(n) \subseteq \text{NSPACE}(n)$ because any Turing machine that operates in time $t(n)$ on every computation branch can use at most $t(n)$ tape cells on every branch. Furthermore $\text{NSPACE}(n) \subseteq \text{SPACE}(n^2)$ due to Savitch's theorem. However, $\text{SPACE}(n^2) \subsetneq \text{SPACE}(n^3)$ because of the space hierarchy theorem. The result follows because $\text{SPACE}(n^3) \subseteq \text{PSPACE}$.

9.7 **(a)** Σ^{500}; **(b)** $(\Sigma \cup \varepsilon)^{500}$; **(c)** $\Sigma^{500}\Sigma^*$; **(d)** $(\Sigma \cup \varepsilon)^{499} \cup \Sigma^{501}\Sigma^*$.

9.15 **(a)** Let A be any language and $k \in \mathcal{N}$. If $A \in \text{P}$, then $pad(A, n^k) \in \text{P}$ because you can determine whether $w \in pad(A, n^k)$ by writing w as $s\#^l$ where s doesn't contain the # symbol, then testing whether $|w| = |s|^k$, and finally testing whether $s \in A$. Implementing the first test in polynomial time is straightforward. The second test runs in time poly($|w|$), and because $|w|$ is poly($|s|$), the test runs in time poly($|s|$) and hence is in polynomial time. If $pad(A, n^k) \in \text{P}$, then $A \in \text{P}$ because you can determine whether $w \in A$ by padding w with # symbols until it has length $|w|^k$ and then testing whether the result is in $pad(A, n^k)$. Both of these tests require only polynomial time.

(b) Assume that P $=$ SPACE(n). Let A be a language in SPACE(n^2) but not in SPACE(n) as shown to exist in the space hierarchy theorem. The language $pad(A, n^2) \in$ SPACE(n) because you have enough space to run the $O(n^2)$ space algorithm for A, using space that is linear in the padded language. Because of the assumption, $pad(A, n^2) \in$ P, hence $A \in$ P by part (a), and hence $A \in$ SPACE(n), due to the assumption once again. But that is a contradiction.

We can use a trapdoor function such as the RSA trapdoor function, to construct a public-key cryptosystem as follows. The public key is the index i generated by the probabilistic machine G. The secret key is the corresponding value t. The encryption algorithm breaks the message m into blocks of size at most $\log N$. For each block w the sender computes f_i. The resulting sequence of strings is the encrypted message. The receiver uses the function h to obtain the original message from its encryption.

EXERCISES

10.1 Show that a circuit family with depth $O(\log n)$ is also a polynomial size circuit family.

10.2 Show that 12 is not pseudoprime because it fails some Fermat test.

10.3 Prove that, if $A \leq_L B$ and B is in NC, then A is in NC.

10.4 Show that the parity function with n inputs can be computed by a branching program that has $O(n)$ nodes.

10.5 Show that the majority function with n inputs can be computed by a branching program that has $O(n^2)$ nodes.

10.6 Show that any function with n inputs can be computed by a branching program that has $O(2^n)$ nodes.

A**10.7** Show that BPP $\subseteq$ PSPACE.

PROBLEMS

10.8 Let A be a regular language over $\{0,1\}$. Show that A has size–depth complexity $(O(n), O(\log n))$.

10.9** A ***Boolean formula is a Boolean circuit wherein every gate has only one output wire. The same input variable may appear in multiple places of a Boolean formula. Prove that a language has a polynomial size family of formulas iff it is in NC^1. Ignore uniformity considerations.

10.10** A ***k-head pushdown automaton (k-PDA) is a deterministic pushdown automaton with k read-only, two-way input heads and a read/write stack. Define the class $\mathrm{PDA}_k = \{A \mid A \text{ is recognized by a } k\text{-PDA}\}$. Show that $\mathrm{P} = \bigcup_k \mathrm{PDA}_k$. (Hint: Recall that P equals alternating log space.)

10.11 Let M be a probabilistic polynomial time Turing machine and let C be a language where, for some fixed $0 < \epsilon_1 < \epsilon_2 < 1$,

a. $w \notin C$ implies $\Pr[M \text{ accepts } w] \leq \epsilon_1$, and
b. $w \in C$ implies $\Pr[M \text{ accepts } w] \geq \epsilon_2$.

Show that $C \in$ BPP. (Hint: Use the result of Lemma 10.5.)

10.12 Show that, if P = NP, then P = PH.

10.13 Show that, if PH = PSPACE, then the polynomial time hierarchy has only finitely many distinct levels.

10.14 Recall that NP^{SAT} is the class of languages that are decided by nondeterministic polynomial time Turing machines with an oracle for the satisfiability problem. Show that $\text{NP}^{SAT} = \Sigma_2\text{P}$.

***10.15** Prove Fermat's little theorem, which is given in Theorem 10.6. (Hint: Consider the sequence $a^1, a^2, \ldots$. What must happen, and how?)

A*10.16 Prove that, for any integer $p > 1$, if p isn't pseudoprime, then p fails the Fermat test for at least half of all numbers in $\mathcal{Z}_p$.

10.17 Prove that, if A is a language in L, a family of branching programs $(B_1, B_2, \ldots)$ exists wherein each B_n accepts exactly the strings in A of length n and is bounded in size by a polynomial in n.

10.18 Prove that, if A is a regular language, a family of branching programs $(B_1, B_2, \ldots)$ exists wherein each B_n accepts exactly the strings in A of length n and is bounded in size by a constant times n.

10.19 Show that, if NP $\subseteq$ BPP then NP = RP.

10.20 Define a ***ZPP-machine*** to be a probabilistic Turing machine which is permitted three types of output on each of its branches: *accept*, *reject*, and *?*. A ZPP-machine M decides a language A if M outputs the correct answer on every input string w (*accept* if $w \in A$ and *reject* if $w \notin A$) with probability at least $\frac{2}{3}$, and M never outputs the wrong answer. On every input, M may output *?* with probability at most $\frac{1}{3}$. Furthermore, the average running time over all branches of M on w must be bounded by a polynomial in the length of w. Show that RP $\cap$ coRP = ZPP.

10.21 Let $EQ_{\text{BP}} = \{\langle B_1, B_2\rangle \mid B_1 \text{ and } B_2 \text{ are equivalent branching programs}\}$. Show that EQ_{BP} is coNP-complete

10.22 Let BPL be the collection of languages that are decided by probabilistic log space Turing machines with error probability $\frac{1}{3}$. Prove that BPL $\subseteq$ P.

SELECTED SOLUTIONS

10.7 If M is a probabilistic TM that runs in polynomial time, we can modify M so that it makes exactly n^r coin tosses on each branch of its computation, for some constant r. Thus the problem of determining the probability that M accepts its input string reduces to counting how many branches are accepting and comparing this number with $\frac{2}{3}2^{(n^r)}$. This count can be performed by using polynomial space.

10.16 Call a a *witness* if it fails the Fermat test for p, that is, if $a^{p-1} \not\equiv 1 \pmod{p}$. Let $\mathcal{Z}_p^*$ be all numbers in $\{1, \ldots, p-1\}$ that are relatively prime to p. If p isn't pseudoprime, it has a witness a in $\mathcal{Z}_p^*$.

Use a to get many more witnesses. Find a unique witness in $\mathcal{Z}_p^*$ for each nonwitness. If $d \in \mathcal{Z}_p^*$ is a nonwitness, you have $d^{p-1} \equiv 1 \pmod{p}$. Hence $da \bmod p \not\equiv 1 \pmod{p}$ and so $da \bmod p$ is a witness. If d_1 and d_2 are distinct nonwitnesses in $\mathcal{Z}_p^*$ then $d_1 a \bmod p \neq d_2 a \bmod p$. Otherwise $(d_1 - d_2)a \equiv 0 \pmod{p}$, and thus $(d_1 - d_2)a = cp$ for some integer c. But d_1 and d_2 are in $\mathcal{Z}_p^*$, and thus $(d_1 - d_2) < p$, so $a = cp/(d_1 - d_2)$ and p have a factor greater than 1 in common, which is impossible because a and p are relatively prime. Thus the number of witnesses in $\mathcal{Z}_p^*$ must be as large as the number of nonwitnesses in $\mathcal{Z}_p^*$ and consequently at least half of the members of $\mathcal{Z}_p^*$ are witnesses.

Next show that every member b of $\mathcal{Z}_p$ that is not relatively prime to p is a witness. If b and p share a factor, then b^e and p share that factor for any $e > 0$. Hence $b^{p-1} \not\equiv 1 \pmod{p}$. Therefore you can conclude that at least half of the member of $\mathcal{Z}_p$ are witnesses.

SELECTED BIBLIOGRAPHY

1. ADLEMAN, L. Two theorems on random polynomial time. In *Proceedings of the Nineteenth IEEE Symposium on Foundations of Computer Science* (1978), pp. 75–83.
2. ADLEMAN, L. M., AND HUANG, M. A. Recognizing primes in random polynomial time. In *Proceedings of the Nineteenth Annual ACM Symposium on the Theory of Computing* (1987), pp. 462–469.
3. ADLEMAN, L. M., POMERANCE, C., AND RUMELY, R. S. On distinguishing prime numbers from composite numbers. *Annals of Mathematics 117* (1983), 173–206.
4. AGRAWAL, M., KAYAL, N., AND SAXENA, N. PRIMES is in P. (2002), `http://www.cse.iitk.ac.in/news/primality.pdf`.
5. AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. *Data Structures and Algorithms*. Addison-Wesley, 1982.
6. AHO, A. V., SETHI, R., AND ULLMAN, J. D. *Compilers: Principles, Techniques, Tools*. Addison-Wesley, 1986.
7. AKL, S. G. *The Design and Analysis of Parallel Algorithms*. Prentice-Hall International, 1989.
8. ALON, N., ERDÖS, P., AND SPENCER, J. H. *The Probabilistic Method*. John Wiley & Sons, 1992.
9. ANGLUIN, D., AND VALIANT, L. G. Fast probabilistic algorithms for Hamiltonian circuits and matchings. *Journal of Computer and System Sciences 18* (1979), 155–193.
10. ARORA, S., LUND, C., MOTWANI, R., SUDAN, M., AND SZEGEDY, M. Proof verification and hardness of approximation problems. In *Proceedings of the Thirty-third IEEE Symposium on Foundations of Computer Science* (1992), pp. 14–23.
11. BAASE, S. *Computer Algorithms: Introduction to Design and Analysis*. Addison-Wesley, 1978.
12. BABAI, L. E-mail and the unexpected power of interaction. In *Proceedings of the Fifth Annual Conference on Structure in Complexity Theory* (1990), pp. 30–44.
13. BACH, E., AND SHALLIT, J. *Algorithmic Number Theory, Vol. 1*. MIT Press, 1996.

14. BALCÁZAR, J. L., DÍAZ, J., AND GABARRÓ, J. *Structural Complexity I, II.* EATCS Monographs on Theoretical Computer Science. Springer Verlag, 1988 (I) and 1990 (II).
15. BEAME, P. W., COOK, S. A., AND HOOVER, H. J. Log depth circuits for division and related problems. *SIAM Journal on Computing 15*, 4 (1986), 994–1003.
16. BLUM, M., CHANDRA, A., AND WEGMAN, M. Equivalence of free boolean graphs can be decided probabilistically in polynomial time. *Information Processing Letters 10* (1980), 80–82.
17. BRASSARD, G., AND BRATLEY, P. *Algorithmics: Theory and Practice.* Prentice-Hall, 1988.
18. CARMICHAEL, R. D. On composite numbers p which satisfy the Fermat congruence $a^{p-1} \equiv p$. *American Mathematical Monthly 19* (1912), 22–27.
19. CHOMSKY, N. Three models for the description of language. *IRE Trans. on Information Theory 2* (1956), 113–124.
20. COBHAM, A. The intrinsic computational difficulty of functions. In *Proceedings of the International Congress for Logic, Methodology, and Philosophy of Science*, Y. Bar-Hillel, Ed. North-Holland, 1964, pp. 24–30.
21. COOK, S. A. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on the Theory of Computing* (1971), pp. 151–158.
22. CORMEN, T., LEISERSON, C., AND RIVEST, R. *Introduction to Algorithms.* MIT Press, 1989.
23. EDMONDS, J. Paths, trees, and flowers. *Canadian Journal of Mathematics 17* (1965), 449–467.
24. ENDERTON, H. B. *A Mathematical Introduction to Logic.* Academic Press, 1972.
25. EVEN, S. *Graph Algorithms.* Pitman, 1979.
26. FELLER, W. *An Introduction to Probability Theory and Its Applications, Vol. 1.* John Wiley & Sons, 1970.
27. FEYNMAN, R. P., HEY, A. J. G., AND ALLEN, R. W. *Feynman lectures on computation.* Addison-Wesley, 1996.
28. GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability—A Guide to the Theory of NP-completeness.* W. H. Freeman, 1979.
29. GILL, J. T. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing 6*, 4 (1977), 675–695.
30. GÖDEL, K. On formally undecidable propositions in *Principia Mathematica* and related systems I. In *The Undecidable*, M. Davis, Ed. Raven Press, 1965, pp. 4–38.
31. GOEMANS, M. X., AND WILLIAMSON, D. P. .878-approximation algorithms for MAX CUT and MAX 2SAT. In *Proceedings of the Twenty-sixth Annual ACM Symposium on the Theory of Computing* (1994), pp. 422–431.
32. GOLDWASSER, S., AND MICALI, S. Probabilistic encryption. *Journal of Computer and System Sciences* (1984), 270–229.

33. GOLDWASSER, S., MICALI, S., AND RACKOFF, C. The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing* (1989), 186–208.

34. GREENLAW, R., HOOVER, H. J., AND RUZZO, W. L. *Limits to Parallel Computation: P-completeness Theory*. Oxford University Press, 1995.

35. HARARY, F. *Graph Theory*, 2d ed. Addison-Wesley, 1971.

36. HARTMANIS, J., AND STEARNS, R. E. On the computational complexity of algorithms. *Transactions of the American Mathematical Society 117* (1965), 285–306.

37. HILBERT, D. Mathematical problems. Lecture delivered before the International Congress of Mathematicians at Paris in 1900. In *Mathematical Developments Arising from Hilbert Problems*, vol. 28. American Mathematical Society, 1976, pp. 1–34.

38. HOFSTADTER, D. R. *Goedel, Escher, Bach: An Eternal Golden Braid.* Basic Books, 1979.

39. HOPCROFT, J. E., AND ULLMAN, J. D. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

40. JOHNSON, D. S. The NP-completeness column: Interactive proof systems for fun and profit. *Journal of Algorithms 9*, 3 (1988), 426–444.

41. KARP, R. M. Reducibility among combinatorial problems. In *Complexity of Computer Computations* (1972), R. E. Miller and J. W. Thatcher, Eds., Plenum Press, pp. 85–103.

42. KARP, R. M., AND LIPTON, R. J. Turing machines that take advice. *ENSEIGN: L'Enseignement Mathematique Revue Internationale 28* (1982).

43. LAWLER, E. L. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1991.

44. LAWLER, E. L., LENSTRA, J. K., RINOOY KAN, A. H. G., AND SHMOYS, D. B. *The Traveling Salesman Problem*. John Wiley & Sons, 1985.

45. LEIGHTON, F. T. *Introduction to Parallel Algorithms and Architectures: Array, Trees, Hypercubes*. Morgan Kaufmann, 1991.

46. LEVIN, L. Universal search problems (in Russian). *Problemy Peredachi Informatsii 9*, 3 (1973), 115–116.

47. LEWIS, H., AND PAPADIMITRIOU, C. *Elements of the Theory of Computation*. Prentice-Hall, 1981.

48. LI, M., AND VITANYI, P. *Introduction to Kolmogorov Complexity and its Applications.* Springer-Verlag, 1993.

49. LICHTENSTEIN, D., AND SIPSER, M. GO is PSPACE hard. *Journal of the ACM* (1980), 393–401.
50. LUBY, M. *Pseudorandomness and Cryptographic Applications.* Princeton University Press, 1996.
51. LUND, C., FORTNOW, L., KARLOFF, H., AND NISAN, N. Algebraic methods for interactive proof systems. *Journal of the ACM 39*, 4 (1992), 859–868.
52. MILLER, G. L. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences 13* (1976), 300–317.
53. NIVEN, I., AND ZUCKERMAN, H. S. *An Introduction to the Theory of Numbers*, 4th ed. John Wiley & Sons, 1980.
54. PAPADIMITRIOU, C. H. *Computational Complexity.* Addison-Wesley, 1994.
55. PAPADIMITRIOU, C. H., AND STEIGLITZ, K. *Combinatorial Optimization (Algorithms and Complexity).* Prentice-Hall, 1982.
56. PAPADIMITRIOU, C. H., AND YANNAKAKIS, M. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences 43*, 3 (1991), 425–440.
57. POMERANCE, C. On the distribution of pseudoprimes. *Mathematics of Computation 37*, 156 (1981), 587–593.
58. PRATT, V. R. Every prime has a succinct certificate. *SIAM Journal on Computing 4*, 3 (1975), 214–220.
59. RABIN, M. O. Probabilistic algorithms. In *Algorithms and Complexity: New Directions and Recent Results*, J. F. Traub, Ed. Academic Press, 1976, pp. 21–39.
60. REINGOLD, O. Undirected st-connectivity in log-space. (2004), `http://www.eccc.uni-trier.de/eccc-reports/2004/TR04-094`.
61. RIVEST, R. L., SHAMIR, A., AND ADLEMAN, L. A method for obtaining digital signatures and public key cryptosytems. *Communications of the ACM 21*, 2 (1978), 120–126.
62. ROCHE, E., AND SCHABES, Y. *Finite-State Language Processing.* MIT Press, 1997.
63. SCHAEFER, T. J. On the complexity of some two-person perfect-information games. *Journal of Computer and System Sciences 16*, 2 (1978), 185–225.
64. SEDGEWICK, R. *Algorithms*, 2d ed. Addison-Wesley, 1989.
65. SHAMIR, A. IP = PSPACE. *Journal of the ACM 39*, 4 (1992), 869–877.
66. SHEN, A. IP = PSPACE: Simplified proof. *Journal of the ACM 39*, 4 (1992), 878–880.
67. SHOR, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing 26*, (1997), 1484–1509.
68. SIPSER, M. Lower bounds on the size of sweeping automata. *Journal of Computer and System Sciences 21*, 2 (1980), 195–202.

69. SIPSER, M. The history and status of the P versus NP question. In *Proceedings of the Twenty-fourth Annual ACM Symposium on the Theory of Computing* (1992), pp. 603–618.

70. STINSON, D. R. *Cryptography: Theory and Practice*. CRC Press, 1995.

71. TARJAN, R. E. *Data structures and network algorithms*, vol. 44 of *CBMS-NSF Reg. Conf. Ser. Appl. Math.* SIAM, 1983.

72. TURING, A. M. On computable numbers, with an application to the Entscheidungsproblem. In *Proceedings, London Mathematical Society*, (1936), pp. 230–265.

73. ULLMAN, J. D., AHO, A. V., AND HOPCROFT, J. E. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

74. VAN LEEUWEN, J., Ed. *Handbook of Theoretical Computer Science A: Algorithms and Complexity*. Elsevier, 1990.

INDEX

Symbols

Introduction to the Theory of
COMPUTATION
Second Edition

This highly anticipated revision of Michael Sipser's popular text builds upon the strengths of the previous edition. It tells the fascinating story of the theory of computation—a subject with beautiful results and exciting unsolved questions at the crossroads of mathematics and computer science. Sipser's candid, crystal-clear style allows students at every level to understand and enjoy this field. His innovative "proof idea" sections reveal the intuition underpinning the formal proofs of theorems by explaining profound concepts in plain English. The new edition incorporates many improvements students and professors have suggested over the years and offers completely updated, classroom-tested problem sets with sample solutions at the end of each chapter.

About the Author

Michael Sipser has taught theoretical computer science and other mathematical subjects at the Massachusetts Institute of Technology for the past 25 years, where he is a Professor of Applied Mathematics and a member of the Computer Science and Artificial Intelligence Laboratory (CSAIL). Currently, he is the head of the Mathematics Department. He enjoys teaching and pondering the many mysteries of complexity theory.

COURSE
●com

THOMSON
COURSE TECHNOLOGY

Thomson Course Technology is part of the Thomson Learning family of companies—dedicated to providing innovative approaches to lifelong learning. Thomson *is* learning.

Visit Thomson Course Technology online at www.course.com

For your lifelong learning needs, www.thomsonlearning.com

90000>
9 780534 950972
ISBN 0-534-95097-3