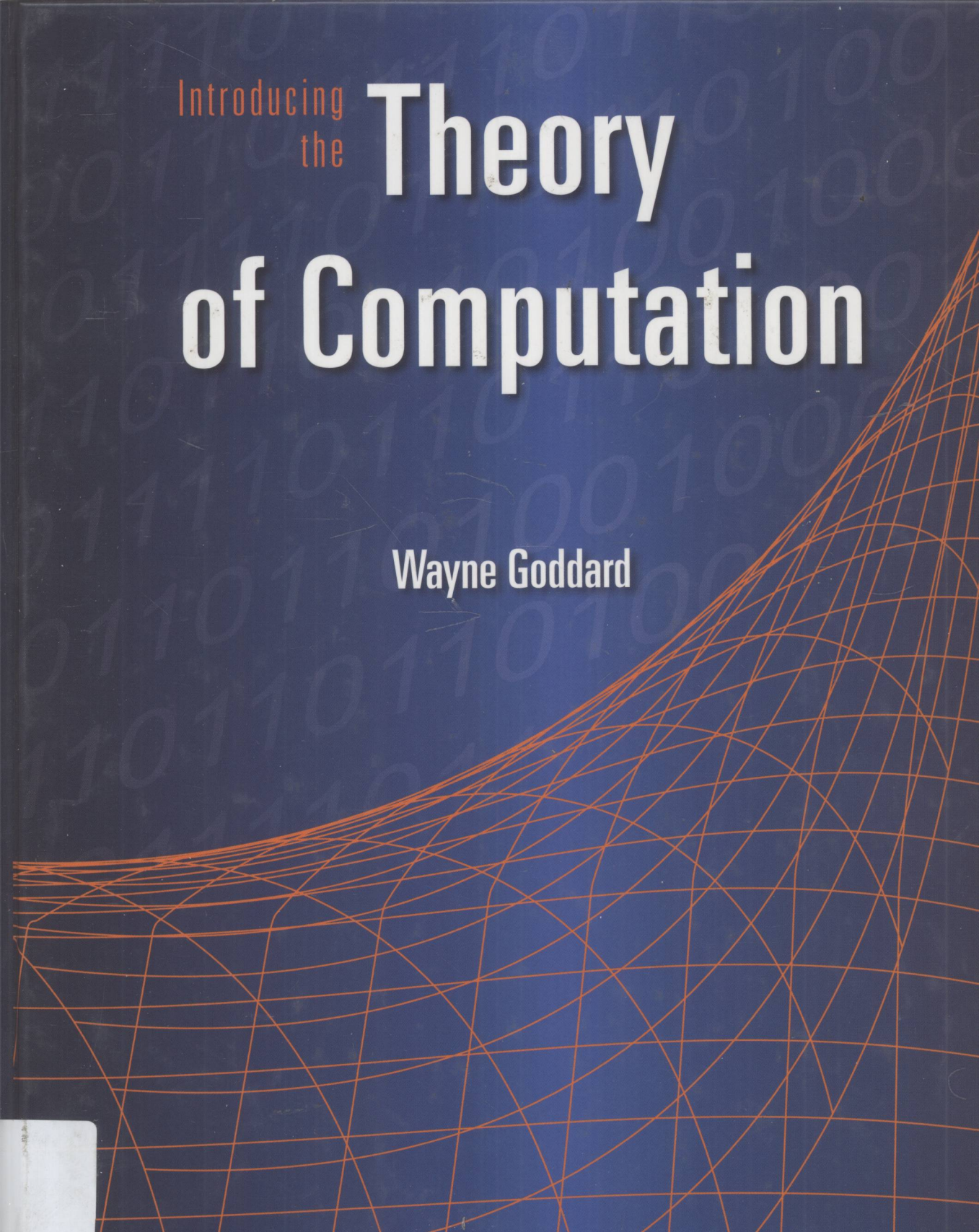
Introducing the Theory of Computation

Wayne Goddard



Introducing the

Theory of Computation

Wayne Goddard
Clemson University

JONES AND BARTLETT PUBLISHERS
Sudbury, Massachusetts
BOSTON TORONTO LONDON SINGAPORE

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

Jones and Bartlett Publishers
Canada
6339 Ormindale Way
Mississauga, Ontario L5V 1J2
Canada

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

Jones and Bartlett's books and products are available through most bookstores and online booksellers. To contact Jones and Bartlett Publishers directly, call 800-832-0034, fax 978-443-8000, or visit our website www.jbpub.com.

Substantial discounts on bulk quantities of Jones and Bartlett's publications are available to corporations, professional associations, and other qualified organizations. For details and specific discount information, contact the special sales department at Jones and Bartlett via the above contact information or send an email to specialsales@jbpub.com.

Production Credits
Acquisitions Editor: Tim Anderson
Production Director: Amy Rose
Editorial Assistant: Melissa Elmore
Senior Marketing Manager: Andrea DeFronzo
Manufacturing Buyer: Therese Connell
Composition: Northeast Compositors
Cover Design: Kate Ternullo
Cover Image: © Cindy Hughes/ShutterStock, Inc. and © Andreas Nilsson/ShutterStock, Inc.
Printing and Binding: Malloy, Inc.
Cover Printing: Malloy, Inc.

Library of Congress Cataloging-in-Publication Data
Goddard, Wayne.
Introducing the theory of computation / Wayne Goddard. — 1st ed.
p. cm.
Includes bibliographical references and index.
ISBN-13: 978-0-7637-4125-9
ISBN-10: 0-7637-4125-6
1. Machine theory. 2. Computational complexity. I. Title.
QA267.G57 2008
511.3'5—dc22
2007049462

6048
Printed in the United States of America
12 11 10 09 08 10 9 8 7 6 5 4 3 2 1

To Stuart, friend and collaborator forever

Preface

Instructions for using this book: Read, think, repeat.

To the Instructor

This is a text for an undergraduate course in the theory of computation; it is also appropriate for courses in automata theory and formal languages. The text covers the standard three models of finite automata, grammars, and Turing machines, as well as undecidability. An introduction to time and space complexity theories and a separate part on the basics of complexity theory are also included.

Goals and Features

When I wrote this text, I focused on the standard material for a course in the theory of computation or automata theory. As a result, this text provides a concise introduction to core topics taught in a course on either of these subjects.

One difference between this text and others on the subject is the use of flowcharts for the pushdown automata. Another difference is that although the material is undoubtedly mathematical, I have tried to reduce the use of mathematical notation. Additionally, this text incorporates the following:

- an engaging, student-friendly writing style that moves through material at a pace appropriate for undergraduate students
- a wide range of problems, varying in level of difficulty, which allows students to test themselves on key material covered in the given chapter

Solutions to selected exercises are in the appendix and are noted with the symbol ★. The most difficult exercises are labeled with the symbol H.

Full solutions are provided in the Online Instructor's Manual, which is available online through the Jones and Bartlett website at: http://www.jbpub.com/catalog/9780763741259/.

Organization

The text is divided into five parts: Regular Languages, Context-Free Languages, Turing Machines, Undecidability, and Complexity Theory. The final chapter in each of the five parts can be viewed as optional—specifically, Chapters 5, 10, 16, and 19. These chapters provide additional information but can be omitted without any impact on the overall course. I include more material than necessary for a single semester course, which provides instructors with the freedom to structure their course and omit or include whichever relevant topics they choose.

To the Student

Welcome to the theory of computation! The material is theoretical, although the early stages are less so. Part IV, Undecidability, is both theoretical and challenging.

As you work through the text, do not lose yourself in the theoretical details. Remember the bigger picture! The finite automata and grammars we see in the first few parts are two of the most efficient and successful techniques in computer programming. The Turing machines and later material show that there are limits to what computer programming can do, even if the actual boundaries are not yet clear. There are problems, procedures, programs, and paradoxes. I encourage you to read and re-read the more difficult sections for better understanding.

The exercises are at varied levels of difficulty. Exercises that are more challenging are marked with the symbol H. Solutions to problems marked with the symbol ★ are provided in the appendix.

Acknowledgments

I was very fortunate to have many people help me write and publish this text. Thank you to my professor at MIT, Michael Sipser, for his instruction. Thanks to family and friends, past and present. In particular, thanks go to Steve Hedetniemi for offering some of his problems and to my students at Clemson University who class-tested early drafts of the text.

Thanks also to the readers and reviewers whose comments greatly improved the book:

Petros Drineas, Rensselaer Polytechnic Institute
Stephen T. Hedetniemi, Clemson University
K. N. King, Georgia State University
Anne-Louise Radimsky, California State University, Sacramento
Neil W. Rickert, Northern Illinois University
R. Duane Skaggs, Morehead State University
Nancy Lynn Tinkham, Rowan University
Jinhui Xu, State University of New York at Buffalo

I want to express my gratitude to the staff at Jones and Bartlett Publishers for their hard work on this text. Thank you to Tim Anderson, Acquisitions Editor; Amy Rose, Production Director; and Melissa Elmore, Editorial Assistant.

Wayne Goddard
Clemson SC

Contents

part

I

Regular Languages

This book is about the fundamental capabilities and ultimate limitations of computation. What can be done with what abilities.

We will see three main models of a computer: a finite automaton, a pushdown automaton, and a Turing machine. In parallel with that we will see other formal ways to describe computation and algorithms, through the language of mathematics, including regular expressions and grammars. In the last two parts of the book, we take our computer and ask what can be solved, and if it can be solved, what resources are required, such as speed and memory. Concepts such as finite automata are certainly useful throughout computer science, but even proving something impossible is good because it tells you where not to look, that you have to compromise on some aspect.

The input to our computers is always strings. We discuss this later, but it is true that everything can be converted to questions about strings.

We start with the simplest form of computer, or maybe, machine. For example, an automatic door. It spends all day either open or closed. The design is simple. Open, closed. Or maybe opening, open, closing, closed. Or maybe there's an override. This is the simplest form of a machine: only internal memory, nothing external, just reacting to events. Ladies and gentlemen, I give you the finite automaton.

Any language is necessarily a finite system applied with different degrees of creativity to an infinite variety of situations, . . . —**David Lodge**

Nature is a self-made machine, more perfectly automated than any automated machine. —**Eric Hoffer**

Mathematics, rightly viewed, possesses not only truth, but supreme beauty—a beauty cold and austere, like that of sculpture, without appeal to any part of our weaker nature, without the gorgeous trappings of painting or music, yet sublimely pure, and capable of a stern perfection such as only the greatest art can show. —**Bertrand Russell**

I hate definitions. —**Benjamin Disraeli**

chapter

1

Finite Automata

The most basic model of a computer is the finite automaton. This is a computer without memory; or rather, the amount of memory is fixed, regardless of the size of the input.

1.1 A Finite Automaton Has States

A **string** is a sequence of characters or **symbols**. A finite-state machine or **finite automaton** (FA) is a device that recognizes a collection of strings. (The plural of automaton is automata.) An FA has three components:

1. An **input tape**, which contains a single string
2. A sensor or **head**, which reads the input string one symbol at a time
3. **Memory**, which can be in any one of a finite number of states—so we speak of the **current state** of the automaton

The "program" of the FA prescribes how the symbols that are read affect the current state. The **final state** for a string is the state the automaton is in when it finishes reading the input.

Operating an FA

1. Set the machine to the start state.
2. If end-of-string then halt.
3. Read a symbol.
4. Update the state according to current state and symbol read.
5. Goto step 2.

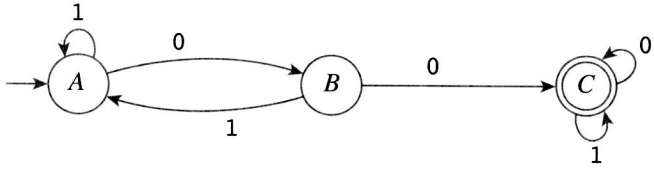
An FA can be described by a diagram. In the diagram, each state is drawn as a circle; we sometimes name a state by putting its name inside the

circle. Each state has, for each symbol, an arrow showing the next state. The initial or **start state** is shown by an arrow into it from no state.

The purpose of an FA is as a **recognizer**—essentially, it acts like a boolean function. For any FA, certain states are designated as **accept** states and the remainder are **reject** states. An **accept state** is indicated by a double circle in the diagram.

Definition An FA **accepts** the input string if the final state is an accept state, otherwise it rejects the input string.

Example 1.1 The following is an FA with 3 states called A, B, and C. The start state is A, and C is the only accept state.



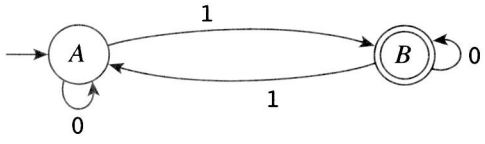
Consider its behavior when the input string is 101001:

Current State	Symbol Read	New State
A	1	A
A	0	B
B	1	A
A	0	B
B	0	C
C	1	C

Here, the final state is C. Similarly, the final state for 11101 is A, and for 0001 it is C.

What does it take to get to the accept state? This machine accepts all strings of 0's and 1's with two consecutive zeroes somewhere.

Example 1.2 Consider the following FA.



This FA accepts strings like `100` and `0101001` and `11111`, and rejects strings like `000` and `0110`.

Can you describe exactly which strings this FA accepts?

If you can't wait, then read on. This FA ignores the symbol `0` (it doesn't change state). It only worries about the symbol `1`; here it alternates states. The first `1` takes it to state B, the second `1` takes it to state A, the third to state B, and so on. So it is in state B whenever an odd number of `1`'s have been read.

That is, this machine accepts all strings of `0`'s and `1`'s with an odd number of `1`'s.

In these examples, we simply use letters for the names of states. Sometimes you can find more descriptive names.

1.2 Building FAs

There is no magic method for building FAs. It takes practice and thinking (though some of the machinery in subsequent chapters will be helpful). In this section, we consider how to build an FA for a specific purpose.

First, we need a few more definitions. An **alphabet** is a set of symbols. A **language** is a set of strings, where the strings have symbols from a specific alphabet. The language of an FA is the set of strings it accepts. For example, the language of the first FA from Example 1.1 is the set of all strings with alphabet $\{0, 1\}$ that contain the substring `00`.

We often use Σ to denote the alphabet. Often the alphabet will be $\{a, b\}$ or $\{0, 1\}$; though this is abusing the term, we refer to strings from the alphabet $\{0, 1\}$ as **binary** strings. A **unary language** is one where the alphabet has only one symbol.

The **length** of a string is the number of symbols in it. The **empty string** has length 0: it is a string without any symbols and is denoted ε.

Sometimes the obvious natural idea works.