

AN INTRODUCTION TO

# FORMAL LANGUAGES AND AUTOMATA

SEVENTH EDITION



JONES & BARTLETT  
LEARNING

**PETER LINZ**

**SUSAN H. RODGER**



World Headquarters  
Jones & Bartlett Learning  
25 Mall Road  
Burlington, MA 01803  
978-443-5000  
info@jblearning.com  
www.jblearning.com

Jones & Bartlett Learning books and products are available through most bookstores and online booksellers. To contact Jones & Bartlett Learning directly, call 800-832-0034, fax 978-443-8000, or visit our website, [www.jblearning.com](http://www.jblearning.com).

Substantial discounts on bulk quantities of Jones & Bartlett Learning publications are available to corporations, professional associations, and other qualified organizations. For details and specific discount information, contact the special sales department at Jones & Bartlett Learning via the above contact information or send an email to [specialsales@jblearning.com](mailto:specialsales@jblearning.com).

Copyright © 2023 by Jones & Bartlett Learning, LLC, an Ascend Learning Company

All rights reserved. No part of the material protected by this copyright may be reproduced or utilized in any form, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without written permission from the copyright owner.

The content, statements, views, and opinions herein are the sole expression of the respective authors and not that of Jones & Bartlett Learning, LLC. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not constitute or imply its endorsement or recommendation by Jones & Bartlett Learning, LLC and such reference shall not be used for advertising or product endorsement purposes. All trademarks displayed are the trademarks of the parties noted herein. *An Introduction to Formal Languages and Automata, Seventh Edition* is an independent publication and has not been authorized, sponsored, or otherwise approved by the owners of the trademarks or service marks referenced in this product.

There may be images in this book that feature models; these models do not necessarily endorse, represent, or participate in the activities represented in the images. Any screenshots in this product are for educational and instructive purposes only. Any individuals and scenarios featured in the case studies throughout this product may be real or fictitious but are used for instructional purposes only.

#### Production Credits

Vice President, Product Management: Marisa R. Urbano	Content Services Manager: Colleen Lamy
Vice President, Product Operations: Christine Emerton	VP, Manufacturing and Inventory Control: Therese Connell
Product Manager: Ned Hinman	Product Fulfillment Manager: Wendy Kilborn
Director, Content Management: Donna Gridley	Composition: Straive
Content Strategist: Melissa Duffy	Project Management: Straive
Content Coordinator: Mark Restuccia	Cover Design: Briana Yates
Director, Project Management and Content Services: Karen Scott	Rights & Permissions Manager: John Rusk
Manager, Project Management: Jessica deMartin	Rights Specialist: James Fortney
Project Manager: Roberta Sherman	Cover and Title Page: © VS148/Shutterstock.
Senior Digital Project Specialist: Angela Dooley	Printing and Binding: Gasch Printing.

#### Library of Congress Cataloging-in-Publication Data

Names: Linz, Peter, author. | Rodger, Susan H., author.  
Title: An introduction to formal languages and automata / Peter Linz, Susan H. Rodger.

Description: Seventh edition. | Burlington, Massachusetts : Jones & Bartlett Learning, [2023] | Includes bibliographical references and index.

Identifiers: LCCN 2021047487 | ISBN 9781284231601 (paperback)

Subjects: LCSH: Formal languages. | Machine theory.

Classification: LCC QA267.3 .L56 2023 | DDC 005.13/1—dc23/eng/20210927

LC record available at <https://lcn.loc.gov/2021047487>

6048

Printed in the United States of America

26 25 24 23 22 10 9 8 7 6 5 4 3 2 1

*To Thomas and the Memory of My Father*

— *S. Rodger*



# CONTENTS

<b>PREFACE</b>	<b>xi</b>
<b>PART I: THEORY</b>	<b>1</b>
<b>1 INTRODUCTION TO THE THEORY OF COMPUTATION</b>	<b>3</b>
1.1 Mathematical Preliminaries and Notation . . . . .	5
Sets . . . . .	5
Functions and Relations . . . . .	8
Graphs and Trees . . . . .	10
Proof Techniques . . . . .	11
1.2 Three Basic Concepts . . . . .	19
Languages . . . . .	19
Grammars . . . . .	22
Automata . . . . .	28
1.3 Some Applications* . . . . .	32
<b>2 FINITE AUTOMATA</b>	<b>39</b>
2.1 Deterministic Finite Accepters . . . . .	40
Deterministic Accepters and Transition Graphs . . . . .	41
Languages and Dfa's . . . . .	43
Regular Languages . . . . .	48
2.2 Nondeterministic Finite Accepters . . . . .	53
Definition of a Nondeterministic Acceptor . . . . .	54
Why Nondeterminism? . . . . .	58

2.3	Equivalence of Deterministic and Nondeterministic Finite Acceptors . . . . .	61
2.4	Reduction of the Number of States in Finite Automata* . . . . .	69
<b>3</b>	<b>REGULAR LANGUAGES AND REGULAR GRAMMARS</b>	<b>77</b>
3.1	Regular Expressions . . . . .	78
	Formal Definition of a Regular Expression . . . . .	78
	Languages Associated with Regular Expressions . . . . .	79
3.2	Connection Between Regular Expressions and Regular Languages . . . . .	84
	Regular Expressions Denote Regular Languages . . . . .	84
	Regular Expressions for Regular Languages . . . . .	87
	Regular Expressions for Describing Simple Patterns . . . . .	93
3.3	Regular Grammars . . . . .	97
	Right- and Left-Linear Grammars . . . . .	97
	Right-Linear Grammars Generate Regular Languages . . . . .	99
	Right-Linear Grammars for Regular Languages . . . . .	101
	Equivalence of Regular Languages and Regular Grammars . . . . .	103
<b>4</b>	<b>PROPERTIES OF REGULAR LANGUAGES</b>	<b>107</b>
4.1	Closure Properties of Regular Languages . . . . .	108
	Closure under Simple Set Operations . . . . .	108
	Closure under Other Operations . . . . .	111
4.2	Elementary Questions about Regular Languages . . . . .	120
4.3	Identifying Nonregular Languages . . . . .	123
	Using the Pigeonhole Principle . . . . .	124
	A Pumping Lemma . . . . .	125
<b>5</b>	<b>CONTEXT-FREE LANGUAGES</b>	<b>137</b>
5.1	Context-Free Grammars . . . . .	138
	Examples of Context-Free Languages . . . . .	139
	Leftmost and Rightmost Derivations . . . . .	141
	Derivation Trees . . . . .	142
	Relation Between Sentential Forms and Derivation Trees . . . . .	145
5.2	Parsing and Ambiguity . . . . .	149
	Parsing and Membership . . . . .	149
	Ambiguity in Grammars and Languages . . . . .	154
5.3	Context-Free Grammars and Programming Languages . . . . .	160
<b>6</b>	<b>SIMPLIFICATION OF CONTEXT-FREE GRAMMARS AND NORMAL FORMS</b>	<b>165</b>
6.1	Methods for Transforming Grammars . . . . .	166
	A Useful Substitution Rule . . . . .	167
	Removing Useless Productions . . . . .	169

	Removing $\lambda$ -Productions . . . . .	173
	Removing Unit-Productions . . . . .	175
6.2	Two Important Normal Forms . . . . .	182
	Chomsky Normal Form . . . . .	182
	Greibach Normal Form . . . . .	185
6.3	A Membership Algorithm for Context-Free Grammars* . . . . .	189
<b>7</b>	<b>PUSHDOWN AUTOMATA</b>	<b>193</b>
7.1	Nondeterministic Pushdown Automata . . . . .	194
	Definition of a Pushdown Automaton . . . . .	195
	The Language Accepted by a Pushdown Automaton . . . . .	198
7.2	Pushdown Automata and Context-Free Languages . . . . .	204
	Pushdown Automata for Context-Free Languages . . . . .	204
	Context-Free Grammars for Pushdown Automata . . . . .	209
7.3	Deterministic Pushdown Automata and Deterministic Context-Free Languages . . . . .	216
7.4	Grammars for Deterministic Context-Free Languages* . . . . .	221
<b>8</b>	<b>PROPERTIES OF CONTEXT-FREE LANGUAGES</b>	<b>227</b>
8.1	Two Pumping Lemmas . . . . .	228
	A Pumping Lemma for Context-Free Languages . . . . .	228
	A Pumping Lemma for Linear Languages . . . . .	233
8.2	Closure Properties and Decision Algorithms for Context-Free Languages . . . . .	237
	Closure of Context-Free Languages . . . . .	238
	Some Decidable Properties of Context-Free Languages . . . . .	242
<b>9</b>	<b>TURING MACHINES</b>	<b>247</b>
9.1	The Standard Turing Machine . . . . .	248
	Definition of a Turing Machine . . . . .	248
	Turing Machines as Language Accepters . . . . .	255
	Turing Machines as Transducers . . . . .	258
9.2	Combining Turing Machines for Complicated Tasks . . . . .	266
9.3	Turing's Thesis . . . . .	271
<b>10</b>	<b>OTHER MODELS OF TURING MACHINES</b>	<b>275</b>
10.1	Minor Variations on the Turing Machine Theme . . . . .	276
	Equivalence of Classes of Automata . . . . .	276
	Turing Machines with a Stay-Option . . . . .	277
	Turing Machines with Semi-Infinite Tape . . . . .	279
	The Off-Line Turing Machine . . . . .	281
10.2	Turing Machines with More Complex Storage . . . . .	284
	Multitape Turing Machines . . . . .	284
	Multidimensional Turing Machines . . . . .	287
10.3	Nondeterministic Turing Machines . . . . .	289

10.4	A Universal Turing Machine . . . . .	292
10.5	Linear Bounded Automata . . . . .	297
<b>11</b>	<b>A HIERARCHY OF FORMAL LANGUAGES AND AUTOMATA</b>	<b>301</b>
11.1	Recursive and Recursively Enumerable Languages . . . . .	302
	Languages That Are Not Recursively Enumerable . . . . .	304
	A Language That Is Not Recursively Enumerable . . . . .	305
	A Language That Is Not Recursively Enumerable but Not Recursive . . . . .	307
11.2	Unrestricted Grammars . . . . .	309
11.3	Context-Sensitive Grammars and Languages . . . . .	315
	Context-Sensitive Languages and Linear Bounded Automata	316
	Relation Between Recursive and Context-Sensitive Languages . . . . .	318
11.4	The Chomsky Hierarchy . . . . .	321
<b>12</b>	<b>LIMITS OF ALGORITHMIC COMPUTATION</b>	<b>325</b>
12.1	Some Problems That Cannot Be Solved by Turing Machines .	326
	Computability and Decidability . . . . .	326
	The Turing Machine Halting Problem . . . . .	327
	Reducing One Undecidable Problem to Another . . . . .	331
12.2	Undecidable Problems for Recursively Enumerable Languages . . . . .	335
12.3	The Post Correspondence Problem . . . . .	339
12.4	Undecidable Problems for Context-Free Languages . . . . .	345
12.5	A Question of Efficiency . . . . .	348
<b>13</b>	<b>OTHER MODELS OF COMPUTATION</b>	<b>351</b>
13.1	Recursive Functions . . . . .	353
	Primitive Recursive Functions . . . . .	354
	Ackermann's Function . . . . .	358
	$\mu$ Recursive Functions . . . . .	359
13.2	Post Systems . . . . .	362
13.3	Rewriting Systems . . . . .	366
	Matrix Grammars . . . . .	366
	Markov Algorithms . . . . .	367
	L-Systems . . . . .	369
<b>14</b>	<b>AN OVERVIEW OF COMPUTATIONAL COMPLEXITY</b>	<b>371</b>
14.1	Efficiency of Computation . . . . .	372
14.2	Turing Machine Models and Complexity . . . . .	374
14.3	Language Families and Complexity Classes . . . . .	378



14.4	The Complexity Classes P and NP . . . . .	381
14.5	Some NP Problems . . . . .	383
14.6	Polynomial-Time Reduction . . . . .	386
14.7	NP-Completeness and an Open Question . . . . .	389
	<b>PART II: APPLICATIONS</b>	<b>393</b>
<b>15</b>	<b>COMPILERS AND PARSING</b>	<b>395</b>
15.1	Compilers . . . . .	396
15.2	Top-Down vs. Bottom-Up Parsing . . . . .	404
15.3	FIRST Function . . . . .	409
15.4	FOLLOW Function . . . . .	416
<b>16</b>	<b>LL PARSING</b>	<b>423</b>
16.1	Context-Free Grammar Conversion to Nondeterministic Pushdown Automaton . . . . .	424
	Pushdown Automata for <i>LL</i> Parsing . . . . .	424
	Algorithm to Convert Context-Free Grammar to NPDA for <i>LL</i> Parsing . . . . .	424
16.2	<i>LL</i> (1) Parse Table . . . . .	429
	<i>LL</i> (1) Parse Table . . . . .	429
16.3	<i>LL</i> (1) Parsing Algorithm . . . . .	431
	<i>LL</i> (1) Parsing Algorithm . . . . .	432
16.4	<i>LL</i> ( <i>k</i> ) Parsing . . . . .	435
<b>17</b>	<b>LR PARSING</b>	<b>441</b>
17.1	Context-Free Grammar Conversion to NPDA . . . . .	443
	Algorithm to Convert Context-Free Grammar to NPDA for <i>LR</i> Parsing . . . . .	443
17.2	Items and Closure . . . . .	449
17.3	DFA Models the <i>LR</i> Parsing Stack . . . . .	453
	Algorithm to Build DFA Modeling LR Parsing . . . . .	454
17.4	<i>LR</i> (1) Parse Table . . . . .	467
	<i>LR</i> (1) Parsing Actions . . . . .	468
	<i>LR</i> (1) Parse Table Algorithm . . . . .	468
17.5	<i>LR</i> (1) Parsing Algorithm . . . . .	475
	<i>LR</i> (1) Parsing Algorithm . . . . .	476
17.6	<i>LR</i> (1) Parsing with $\lambda$ -Productions . . . . .	481
17.7	<i>LR</i> (1) Parsing Conflicts . . . . .	492
	<b>APPENDIX A FINITE-STATE TRANSDUCERS</b>	<b>499</b>
A.1	A General Framework . . . . .	499
A.2	Mealy Machines . . . . .	500
A.3	Moore Machines . . . . .	502
A.4	Moore and Mealy Machine Equivalence . . . . .	504

A.5 Mealy Machine Minimization . . . . .	508
A.6 Moore Machine Minimization . . . . .	513
A.7 Limitations of Finite-State Transducers . . . . .	514
<b>APPENDIX B JFLAP: A USEFUL TOOL</b>	<b>517</b>
<b>ANSWERS: SOLUTIONS AND HINTS FOR SELECTED EXERCISES</b>	<b>519</b>
<b>REFERENCES FOR FURTHER READING</b>	<b>573</b>
<b>INDEX</b>	<b>575</b>

# PREFACE

This book is designed for an introductory course on formal languages, automata, computability, and related matters. These topics form a major part of what is known as the theory of computation. A course on this subject matter is now standard in the computer science curriculum and is often taught fairly early in the program. Hence, the prospective audience for this book consists primarily of sophomores and juniors majoring in computer science or computer engineering.

Prerequisites for the material in this book are a knowledge of some higher-level programming language (commonly C, C++, Python™, or Java™) and familiarity with the fundamentals of data structures and algorithms. A course in discrete mathematics that includes set theory, functions, relations, logic, and elements of mathematical reasoning is essential. Such a course is part of the standard introductory computer science curriculum.

The study of the theory of computation has several purposes, most importantly (1) to familiarize students with the foundations and principles of computer science, (2) to teach material that is useful in subsequent courses, and (3) to strengthen students' ability to carry out formal and rigorous mathematical arguments. The presentation I have chosen for this text favors the first two purposes, although I would argue that it also serves the third. To present ideas clearly and to give students insight into the material, the text stresses intuitive motivation and illustration of ideas through examples. When there is a choice, I prefer arguments that are easily grasped to those that are concise and elegant but difficult in concept. I state definitions and theorems precisely and give the motivation for proofs

but often leave out the routine and tedious details. I believe that this is desirable for pedagogical reasons. Many proofs are unexciting applications of induction or contradiction with differences that are specific to particular problems. Presenting such arguments in full detail is not only unnecessary, but it interferes with the flow of the story. Therefore, quite a few of the proofs are brief, and someone who insists on completeness may consider them lacking in detail. I do not see this as a drawback. Mathematical skills are not the by-product of reading someone else's arguments, but they come from thinking about the essence of a problem, discovering ideas suitable to make the point, then carrying them out in precise detail. The latter skill certainly has to be learned, and I think that the proof sketches in this text provide very appropriate starting points for such a practice.

Computer science students sometimes view a course in the theory of computation as unnecessarily abstract and of no practical consequence. To convince them otherwise, one needs to appeal to their specific interests and strengths, such as tenacity and inventiveness in dealing with hard-to-solve problems. Because of this, my approach emphasizes learning through problem solving.

By a problem-solving approach, I mean that students learn the material primarily through problem-type illustrative examples that show the motivation behind the concepts, as well as their connection to the theorems and definitions. At the same time, the examples may involve a nontrivial aspect, for which students must discover a solution. In such an approach, homework exercises contribute to a major part of the learning process. The exercises at the end of each section are designed to illuminate and illustrate the material and call on students' problem-solving ability at various levels. Some of the exercises are fairly simple, picking up where the discussion in the text leaves off and asking students to carry on for another step or two. Other exercises are very difficult, challenging even the best minds. A good mix of such exercises can be a very effective teaching tool. Students need not be asked to solve all problems, but should be assigned those that support the goals of the course and the viewpoint of the instructor. Computer science curricula differ from institution to institution; while a few emphasize the theoretical side, others are almost entirely oriented toward practical application. I believe that this text can serve either of these extremes, provided that the exercises are selected carefully with the students' background and interests in mind. At the same time, the instructor needs to inform the students about the level of abstraction that is expected of them. This is particularly true of the proof-oriented exercises. When I say "prove that" or "show that," I have in mind that the student should think about how a proof can be constructed and then produce a clear argument. How formal such a proof should be needs to be determined by the instructor, and students should be given guidelines on this early in the course.

The content of the text is appropriate for a one-semester course. Most of the material can be covered, although some choice of emphasis will have

to be made. In my classes, I generally gloss over proofs, giving just enough coverage to make the result plausible, and then ask students to read the rest on their own. Overall, though, little can be skipped entirely without potential difficulties later on. A few sections, which are marked with an asterisk, can be omitted without loss to later material. Most of the material, however, is essential and must be covered.

Appendix B briefly introduces JFLAP, an interactive software tool available for free at [www.jflap.org](http://www.jflap.org) that is of great help in both learning and teaching the material in this book. It aids in understanding the concepts and is a great time saver in the actual construction of the solutions to the exercises. I highly recommend incorporating JFLAP into the course.

The seventh edition of this book features two additions. The first is a large number of new exercises, collected at the chapter ends, under the heading Introductory Exercises to distinguish them from the already existing Exercises. These new exercises are largely very simple and require only an understanding of the concepts. They are intended as a bridge to the often much more difficult Exercises. The instructor can decide where and for whom these new exercises can be of help. Chapters 1–14 of the sixth edition, with the new exercises, are now reorganized as Part I: Theory.

Substantial new material comes in Part II: Applications, where we discuss some of the important issues in compiler construction in the context of the material in Part I. This new material comes in response to perennial student questions such as “How does this theory apply to the mostly practical matters of computer science?” or even “Why do we have to know all this abstract stuff?” We hope we have given a satisfactory response to such questions.

How can this material be integrated into an existing course? Instructors who cannot find the time for it might use Part II as suggested reading for curious students. But the material is important and attractive to many students, so we recommend that some attempt to cover it in class be made. Part I introduces the main aspects of the theory of computation, but not everything there is essential. Not only the starred sections, but many sections in the latter part can be treated lightly or skipped altogether without loss of continuity. Doing so might free a week or so for Part II. This coverage can benefit many students. Those who never take a course on compilers will get some understanding of what is involved in compiler design; for others who do, it provides a first look at what will be covered later in more detail.

Peter Linz  
Susan H. Rodger

