An Introduction to Formal Languages and Automata

Third Edition

Peter Linz
University of California at Davis

JONES AND BARTLETT PUBLISHERS
Sudbury, Massachusetts
BOSTON TORONTO LONDON SINGAPORE

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

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

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

Library of Congress Cataloging-in-Publication Data

Linz, Peter.
An introduction to formal languages and automata / Peter Linz.--3rd ed.
p. cm.
Includes bibliographical references and index.
ISBN 0-7637-1422-4
1. Formal languages. 2. Machine theory. I. Title.

QA267.3 .L56 2000
511.3--dc21 00-062546

Chief Executive Officer: Clayton Jones
Chief Operating Officer: Don W. Jones, Jr.
Executive Vice President and Publisher: Tom Manning
V.P., Managing Editor: Judith H. Hauck
V.P., College Editorial Director: Brian L. McKean
V.P., Design and Production: Anne Spencer
V.P., Sales and Marketing: Paul Shepardson
V.P., Manufacturing and Inventory Control: Therese Bräuer
Senior Acquisitions Editor: Michael Stranz
Development and Product Manager: Amy Rose
Marketing Director: Jennifer Jacobson
Production Coordination: Trillium Project Management
Cover Design: Night & Day Design
Composition: Northeast Compositors
Printing and Binding: Courier Westford
Cover printing: John Pow Company, Inc.

Cover Image © Jim Wehtje

This book was typeset in Textures 2.1 on a Macintosh G4. The font families used were Computer Modern, Optima, and Futura. The first printing was printed on 50 lb. Decision 94 Opaque.

Printed in the United States of America

04 03 02 01 10 9 8 7 6 5 4 3 2

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++, 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 interferes with the flow of the story. Therefore, quite a few of the proofs are sketchy 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 byproduct of reading someone else's arguments, but 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.

Students in computer science sometimes view a course in the theory of computation as unnecessarily abstract and of little 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. To help instructors, I have provided separately an instructor's guide that outlines the solutions of the exercises and suggests their pedagogical value. Students need not be asked to solve all problems but should be assigned those which 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 might 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, skimpy as they are in the text. I usually give just enough coverage to make the result plausible, asking 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.

The first edition of this book was published in 1990, the second appeared in 1996. The need for yet another edition is gratifying and indicates that my approach, via languages rather than computations, is still viable. The changes for the second edition were evolutionary rather than revolutionary and addressed the inevitable inaccuracies and obscurities of the first edition. It seems, however, that the second edition had reached a point of stability that requires few changes, so the bulk of the third edition is identical to the previous one. The major new feature of the third edition is the inclusion of a set of solved exercises.

Initially, I felt that giving solutions to exercises was undesirable because it limited the number of problems that can be assigned for homework. However, over the years I have received so many requests for assistance from students everywhere that I concluded that it is time to relent. In this edition I have included solutions to a small number of exercises. I have also added some new exercises to keep from reducing the unsolved problems too much. In selecting exercises for solution, I have favored those that have significant instructional values. For this reason, I give not only the answers, but show the reasoning that is the basis for the final result. Many exercises have the same theme; often I choose a representative case to solve, hoping that a student who can follow the reasoning will be able to transfer it to a set of similar instances. I believe that solutions to a carefully selected set of exercises can help students increase their problem-solving skills and still leave instructors a good set of unsolved exercises. In the text, exercises for which a solution or a hint is given are identified with ●.

Also in response to suggestions, I have identified some of the harder exercises. This is not always easy, since the exercises span a spectrum of difficulty and because a problem that seems easy to one student may give considerable trouble to another. But there are some exercises that have posed a challenge for a majority of my students. These are marked with a single star (★). There are also a few exercises that are different from most in that they have no clear-cut answer. They may call for speculation, suggest additional reading, or require some computer programming. While they are not suitable for routine homework assignment, they can serve as entry points for further study. Such exercises are marked with a double star (★★).

Over the last ten years I have received helpful suggestions from numerous reviewers, instructors, and students. While there are too many individuals to mention by name, I am grateful to all of them. Their feedback has been invaluable in my attempts to improve the text.

Peter Linz

Contents