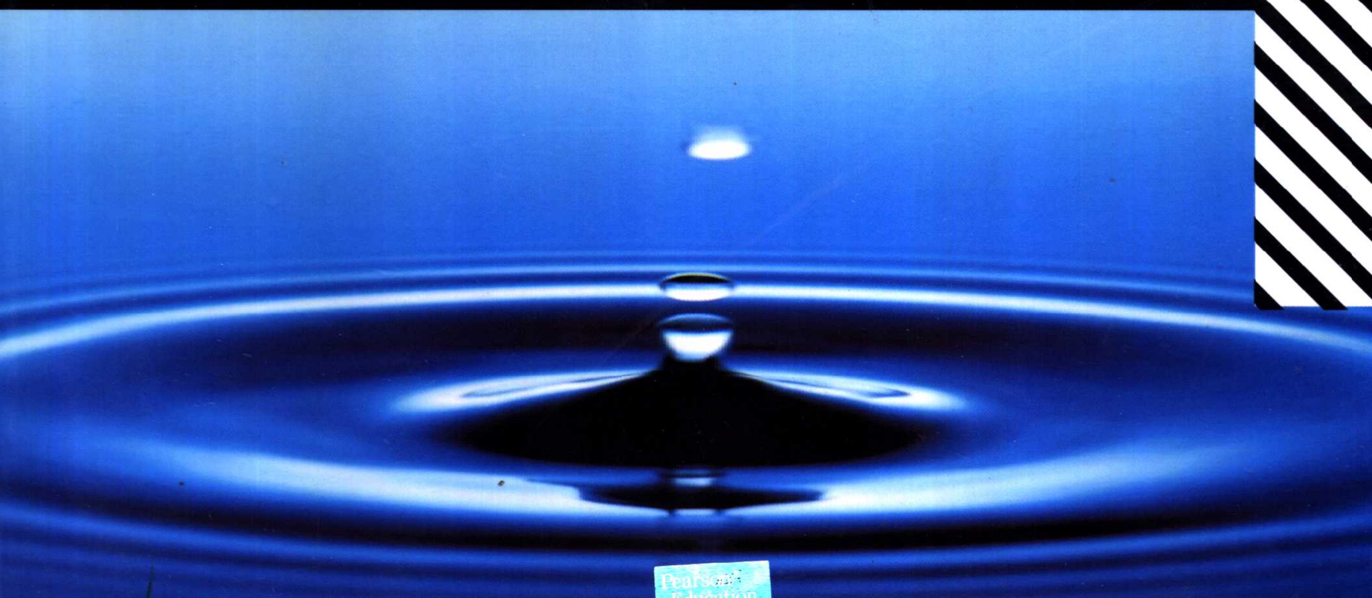
大学计算机教育国外著名教材、教参系列（影印版）

Introduction to Automata Theory, Languages, and Computation

(Second Edition)

John E. Hopcroft
Rajeev Motwani
Jeffrey D. Ullman

自动机理论、语言和计算导论（第2版）



清华大学出版社
http://www.tup.tsinghua.edu.cn

培生教育出版集团
http://www.pearsoned.com

Introduction to Automata Theory, Languages, and Computation

SECOND EDITION

自动机理论、语言和计算导论

第 2 版

JOHN E. HOPCROFT

Cornell University

RAJEEV MOTWANI

Stanford University

JEFFREY D. ULLMAN

Stanford University

清华大学出版社　　培生教育出版集团

（京）新登字 158 号

Introduction to Automata Theory, Languages, and Computation 2nd ed.
John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman

北京市版权局著作权合同登记号：图字：01-2000-01-0112

书　名：Introduction to Automata Theory, Languages, and Computation 2nd ed.
作　者：John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman
出版者：清华大学出版社（北京清华大学学研大厦，邮编 100084）
　　　　http:// www.tup.tsinghua.edu.cn
印刷者：北京密云胶印厂
发行者：新华书店总店北京发行所
开　本：787×960 1/16　印张：33.75
版　次：2002 年 6 月第 1 版　2002 年 6 月第 1 次印刷
书　号：ISBN 7-302-05021-X/TP • 2926
印　数：0001～5000
定　价：47.00 元

出 版 说 明

进入 21 世纪，世界各国的经济、科技以及综合国力的竞争将更加激烈。竞争的中心无疑是对人才的争夺。谁拥有大量高素质的人才，谁就能在竞争中取得优势。高等教育，作为培养高素质人才的事业，必然受到高度重视。目前我国高等教育的教材更新较慢，为了加快教材的更新频率，教育部正在大力促进我国高校采用国外原版教材。

清华大学出版社从 1996 年开始，与国外著名出版公司合作，影印出版了“大学计算机教育丛书（影印版）”等一系列引进图书，受到了国内读者的欢迎和支持。跨入 21 世纪，我们本着为我国高等教育教材建设服务的初衷，在已有的基础上，进一步扩大选题内容，改变图书开本尺寸，一如既往地请有关专家挑选适用于我国高校本科及研究生计算机教育的国外经典教材或著名教材以及教学参考书，组成本套“大学计算机教育国外著名教材、教参系列（影印版）”，以飨读者。深切期盼读者及时将使用本系列教材、教参的效果和意见反馈给我们。更希望国内专家、教授积极向我们推荐国外计算机教育的优秀教材，以利我们把“大学计算机教育国外著名教材、教参系列（影印版）”做得更好，更适合高校师生的需要。

计算机引进版图书编辑室

2002.3

大学计算机教育国外著名教材、教参系列
（影印版）图书目录

1. UNIX Network Programming Vol. 2: Interprocess Communications 2nd *ed.* 1999/*W. Richard Stevens*（UNIX 网络编程卷 2：进程间通信 第 2 版 580 页）
2. The 80x86 IBM PC and Compatible Computers（Volumes Ⅰ & Ⅱ）Assembly Language, Design, and Interfacing 3rd *ed.* 2000/*Muhammed Ali Mazidi, Janice Gillispie Mazidi*（80x86 IBM PC 及兼容计算机卷Ⅰ和卷Ⅱ：汇编语言、设计与接口技术 第 3 版 1020 页）
3. Principles of Distributed Database Systems 2nd *ed.* 1999/*M. Tamer Özsu, Patrick Valduriez*（分布式数据库系统原理 第 2 版 688 页）
4. Network Security Essentials: Applications and Standards 2000/ *William Stallings*（网络安全基础教程 382 页）
5. Cryptography and Network Security: Principles and Practice 2nd *ed.* 1999/*William Stallings*（密码学与网络安全：原理与实践 第 2 版 590 页）
6. Data Structures and Algorithm Analysis in C++ 2nd *ed.* 1999/*Mark Allen Weiss*（数据结构与算法分析 C++描述 第 2 版 606 页）
7. PCI-X System Architecture 2001/*Tom Shanley*（PCI-X 系统的体系结构 734 页）
8. Introduction to Automata Theory, Languages, and Computation 2nd *ed.* 2001/*John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman*（自动机理论、语言和计算导论 第 2 版 535 页）
9. Operating Systems: A Systematic View 5th *ed.* 2001/*William S. Davis, T. M. Rajkumar*（操作系统实践与应用 第 5 版 636 页）
10. Introduction to Logic Design 2002/*Alan B. Marcovitz*（逻辑设计基础 584 页）

Preface

In the preface from the 1979 predecessor to this book, Hopcroft and Ullman marveled at the fact that the subject of automata had exploded, compared with its state at the time they wrote their first book, in 1969. Truly, the 1979 book contained many topics not found in the earlier work and was about twice its size. If you compare this book with the 1979 book, you will find that, like the automobiles of the 1970's, this book is "larger on the outside, but smaller on the inside." That sounds like a retrograde step, but we are happy with the changes for several reasons.

First, in 1979, automata and language theory was still an area of active research. A purpose of that book was to encourage mathematically inclined students to make new contributions to the field. Today, there is little direct research in automata theory (as opposed to its applications), and thus little motivation for us to retain the succinct, highly mathematical tone of the 1979 book.

Second, the role of automata and language theory has changed over the past two decades. In 1979, automata was largely a graduate-level subject, and we imagined our reader was an advanced graduate student, especially those using the later chapters of the book. Today, the subject is a staple of the undergraduate curriculum. As such, the content of the book must assume less in the way of prerequisites from the student, and therefore must provide more of the background and details of arguments than did the earlier book.

A third change in the environment is that Computer Science has grown to an almost unimaginable degree in the past two decades. While in 1979 it was often a challenge to fill up a curriculum with material that we felt would survive the next wave of technology, today very many subdisciplines compete for the limited amount of space in the undergraduate curriculum.

Fourthly, CS has become a more vocational subject, and there is a severe pragmatism among many of its students. We continue to believe that aspects of automata theory are essential tools in a variety of new disciplines, and we believe that the theoretical, mind-expanding exercises embodied in the typical automata course retain their value, no matter how much the student prefers to learn only the most immediately monetizable technology. However, to assure a continued place for the subject on the menu of topics available to the computer science student, we believe it is necessary to emphasize the applications along with the mathematics. Thus, we have replaced a number of the more abstruse topics in the earlier book with examples of how the ideas are used today. While applications of automata and language theory to compilers are now so well understood that they are normally covered in a compiler course, there are a variety of more recent uses, including model-checking algorithms to verify protocols and document-description languages that are patterned on context-free grammars.

A final explanation for the simultaneous growth and shrinkage of the book is that we were today able to take advantage of the TeX and LaTeX typesetting systems developed by Don Knuth and Les Lamport. The latter, especially, encourages the "open" style of typesetting that makes books larger, but easier to read. We appreciate the efforts of both men.

Use of the Book

This book is suitable for a quarter or semester course at the Junior level or above. At Stanford, we have used the notes in CS154, the course in automata and language theory. It is a one-quarter course, which both Rajeev and Jeff have taught. Because of the limited time available, Chapter 11 is not covered, and some of the later material, such as the more difficult polynomial-time reductions in Section 10.4 are omitted as well. The book's Web site (see below) includes notes and syllabi for several offerings of CS154.

Some years ago, we found that many graduate students came to Stanford with a course in automata theory that did not include the theory of intractability. As the Stanford faculty believes that these ideas are essential for every computer scientist to know at more than the level of "NP-complete means it takes too long," there is another course, CS154N, that students may take to cover only Chapters 8, 9, and 10. They actually participate in roughly the last third of CS154 to fulfill the CS154N requirement. Even today, we find several students each quarter availing themselves of this option. Since it requires little extra effort, we recommend the approach.

Prerequisites

To make best use of this book, students should have taken previously a course covering discrete mathematics, e.g., graphs, trees, logic, and proof techniques. We assume also that they have had several courses in programming, and are familiar with common data structures, recursion, and the role of major system components such as compilers. These prerequisites should be obtained in a typical freshman-sophomore CS program.

Exercises

The book contains extensive exercises, with some for almost every section. We indicate harder exercises or parts of exercises with an exclamation point. The hardest exercises have a double exclamation point.

Some of the exercises or parts are marked with a star. For these exercises, we shall endeavor to maintain solutions accessible through the book's Web page. These solutions are publicly available and should be used for self-testing. Note that in a few cases, one exercise B asks for modification or adaptation of your solution to another exercise A. If certain parts of A have solutions, then you should expect the corresponding parts of B to have solutions as well.

Support on the World Wide Web

The book's home page is

```
http://www-db.stanford.edu/~ullman/ialc.html
```

Here are solutions to starred exercises, errata as we learn of them, and backup materials. We hope to make available the notes for each offering of CS154 as we teach it, including homeworks, solutions, and exams.

Acknowledgements

A handout on "how to do proofs" by Craig Silverstein influenced some of the material in Chapter 1. Comments and errata on drafts of this book were received from: Zoe Abrams, George Candea, Haowen Chen, Byong-Gun Chun, Jeffrey Shallit, Bret Taylor, Jason Townsend, and Erik Uzureau. They are gratefully acknowledged. Remaining errors are ours, of course.

J. E. H.
R. M.
J. D. U.
Ithaca NY and Stanford CA
September, 2000

Table of Contents

Chapter 1

Automata: The Methods and the Madness

Automata theory is the study of abstract computing devices, or "machines." Before there were computers, in the 1930's, A. Turing studied an abstract machine that had all the capabilities of today's computers, at least as far as in what they could compute. Turing's goal was to describe precisely the boundary between what a computing machine could do and what it could not do; his conclusions apply not only to his abstract *Turing machines*, but to today's real machines.

In the 1940's and 1950's, simpler kinds of machines, which we today call "finite automata," were studied by a number of researchers. These automata, originally proposed to model brain function, turned out to be extremely useful for a variety of other purposes, which we shall mention in Section 1.1. Also in the late 1950's, the linguist N. Chomsky began the study of formal "grammars." While not strictly machines, these grammars have close relationships to abstract automata and serve today as the basis of some important software components, including parts of compilers.

In 1969, S. Cook extended Turing's study of what could and what could not be computed. Cook was able to separate those problems that can be solved efficiently by computer from those problems that can in principle be solved, but in practice take so much time that computers are useless for all but very small instances of the problem. The latter class of problems is called "intractable," or "NP-hard." It is highly unlikely that even the exponential improvement in computing speed that computer hardware has been following ("Moore's Law") will have significant impact on our ability to solve large instances of intractable problems.

All of these theoretical developments bear directly on what computer scientists do today. Some of the concepts, like finite automata and certain kinds of formal grammars, are used in the design and construction of important kinds of software. Other concepts, like the Turing machine, help us understand what