大学计算机教育国外著名教材系列（影印版）

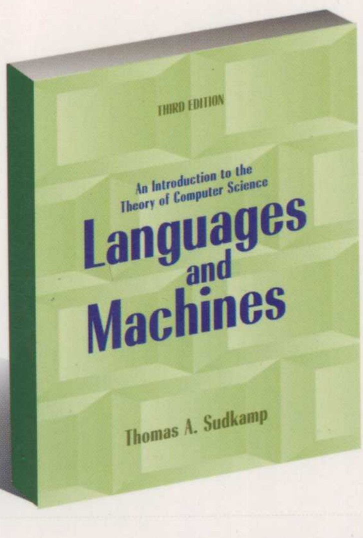
# LANGUAGES AND MACHINES

## AN INTRODUCTION TO THE THEORY OF COMPUTER SCIENCE Third Edition

# 语言与机器（第3版）

## 计算机科学理论导论

Thomas A. Sudkamp 著




清华大学出版社

大学计算机教育国外著名教材系列（影印版）

# Languages and Machines

## An Introduction to the Theory of Computer Science

Third Edition

# 语言与机器

## 计算机科学理论导论

（第 3 版）



Thomas A. Sudkamp

清华大学出版社
北　京

北京市版权局著作权合同登记号 图字 01-2005-3987 号



图书在版编目(CIP)数据

语言与机器：计算机科学理论导论：第 3 版=Languages and Machines, 3e：英文 / （美） 苏达坎（Sudkamp, T.A.）著. —北京：清华大学出版社，2007.7
ISBN 978-7-302-15172-2

I. 语… II. 苏… III. 计算机科学—英文 IV. TP3

中国版本图书馆 CIP 数据核字（2007）第 069504 号

**责任编辑：**龙啟铭
**责任印制：**孟凡玉

**出 版 者：**清华大学出版社
http://www.tup.com.cn
**社总机：**010-6277 0175
**地　　址：**北京清华大学学研大厦
**邮　　编：**100084
**客户服务：**010-6277 6969

**印 刷 者：**北京市昌平环球印刷厂
**装 订 者：**三河市金元印装有限公司
**发 行 者：**新华书店总店北京发行所
**开　　本：**185×230 **印张：**42.5
**版　　次：**2007 年 7 月第 1 版 2007 年 7 月第 1 次印刷
**印　　数：**1~3000
**定　　价：**69.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：010-62770177 转 3103 产品编号：019033-01

$\langle dedication \rangle \rightarrow \langle parents \rangle$

$\langle parents \rangle \rightarrow \langle first\ name \rangle\ \langle last\ name \rangle$

$\langle first\ name \rangle \rightarrow Donald \mid Mary$

$\langle last\ name \rangle \rightarrow Sudkamp$

# 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竞争。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，组成本套“大学计算机教育国外著名教材系列（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系列（影印版）”做得更好，更适合高校师生的需要。

清华大学出版社

# Preface

The objective of the third edition of *Languages and Machines: An Introduction to the Theory of Computer Science* remains the same as that of the first two editions, to provide a mathematically sound presentation of the theory of computer science at a level suitable for junior- and senior-level computer science majors. The impetus for the third edition was threefold: to enhance the presentation by providing additional motivation and examples; to expand the selection of topics, particularly in the area of computational complexity; and to provide additional flexibility to the instructor in the design of an introductory course in the theory of computer science.

While many applications-oriented students question the importance of studying theoretical foundations, it is this subject that addresses the "big picture" issues of computer science. When today's programming languages and computer architectures are obsolete and solutions have been found for problems currently of interest, the questions considered in this book will still be relevant. What types of patterns can be algorithmically detected? How can languages be formally defined and analyzed? What are the inherent capabilities and limitations of algorithmic computation? What problems have solutions that require so much time or memory that they are realistically intractable? How do we compare the relative difficulty of two problems? Each of these questions will be addressed in this text.

## Organization

Since most computer science students at the undergraduate level have little or no background in abstract mathematics, the presentation is intended not only to introduce the foundations of computer science but also to increase the student's mathematical sophistication. This is accomplished by a rigorous presentation of the concepts and theorems of the subject accompanied by a generous supply of examples. Each chapter ends with a set of exercises that reinforces and augments the material covered in the chapter.

To make the topics accessible, no special mathematical prerequisites are assumed. Instead, Chapter 1 introduces the mathematical tools of the theory of computing: naive set theory, recursive definitions, and proof by mathematical induction. With the exception of the specialized topics in Sections 1.3 and 1.4, Chapters 1 and 2 provide background material that will be used throughout the text. Section 1.3 introduces cardinality and diagonalization, which are used in the counting arguments that establish the existence of undecidable languages and uncomputable functions. Section 1.4 examines the use of self-reference in proofs by contradiction. This technique is used in undecidability proofs, including the proof that there is no solution to the Halting Problem. For students who have completed a course in discrete mathematics, most of the material in Chapter 1 can be treated as review.

Recognizing that courses in the foundations of computing may emphasize different topics, the presentation and prerequisite structure of this book have been designed to permit a course to investigate particular topics in depth while providing the ability to augment the primary topics with material that introduces and explores the breadth of computer science theory. The core material for courses that focus on a classical presentation of formal and automata language theory, on computability and undecidability, on computational complexity, and on formal languages as the foundation for programming language definition and compiler design are given in the following table. A star next to a section indicates that the section may be omitted without affecting the continuity of the presentation. A starred section usually contains the presentation of an application, the introduction of a related topic, or a detailed proof of an advanced result in the subject.

| Formal Language and Automata Theory | Computability Theory | Computational Complexity | Formal Languages for Programming Languages |
|---|---|---|---|
| Chap. 1: 1–3, 6–8 | Chap. 1: all | Chap. 1: all | Chap. 1: 1–3, 6–8 |
| Chap. 2: 1–3, 4* | Chap. 2: 1–3, 4* | Chap. 2: 1–3, 4* | Chap. 2: 1–4 |
| Chap. 3: 1–3, 4* | Chap. 5: 1–6, 7* | Chap. 5: 1–4, 5–7* | Chap. 3: 1–4 |
| Chap. 4: 1–5, 6*, 7 | Chap. 8: 1–7, 8* | Chap. 8: 1–7, 8* | Chap. 4: 1–5, 6*, 7 |
| Chap. 5: 1–6, 7* | Chap. 9: 1–5, 6* | Chap. 9: 1–4, 5–6* | Chap. 5: 1–6, 7* |
| Chap. 6: 1–5, 6* | Chap. 10: 1 | Chap. 11: 1–4, 5* | Chap. 7: 1–3, 4–5* |
| Chap. 7: 1–5 | Chap. 11: all | Chap. 14: 1–4, 5–7* | Chap. 18: all |
| Chap. 8: 1–7, 8* | Chap. 12: all | Chap. 15: all | Chap. 19: all |
| Chap. 9: 1–5, 6* | Chap. 13: all | Chap. 16: 1–6, 7* | Chap. 20: all |
| Chap. 10: all | | Chap. 17: all | |

The classical presentation of formal language and automata theory examines the relationships between the grammars and abstract machines of the Chomsky hierarchy. The computational properties of deterministic finite automata, pushdown automata, linear-bounded automata, and Turing machines are examined. The analysis of the computational power of abstract machines culminates by establishing the equivalence of language recognition by Turing machines and language generation by unrestricted grammars.

Computability theory examines the capabilities and limitations of algorithmic problem solving. The coverage of computability includes decidability and the Church-Turing Thesis, which is supported by the establishment of the equivalence of Turing computability and $\mu$-recursive functions. A diagonalization argument is used to show that the Halting Problem for Turing machines is unsolvable. Problem reduction is then used to establish the undecidability of a number of questions on the capabilities of algorithmic computation.

The study of computational complexity begins by considering methods for measuring the resources required by a computation. The Turing machine is selected as the framework for the assessment of complexity, and time and space complexity are measured by the number of transitions and amount of memory used in Turing machine computations. The class $\mathcal{P}$ of problems that are solvable by deterministic Turing machines in polynomial time is identified as the set problems that have efficient algorithmic solutions. The class $\mathcal{NP}$ and the theory of NP-completeness are then introduced. Approximation algorithms are used to obtain near-optimal solutions for NP-complete optimization problems.

The most important application of formal language theory to computer science is the use of grammars to specify the syntax of programming languages. A course with the focus of using formal techniques to define programming languages and develop efficient parsing strategies begins with the introduction of context-free grammars to generate languages and finite automata to recognize patterns. After the introduction to language definition, Chapters 18–20 examine the properties of LL and LR grammars and deterministic parsing of languages defined by these types of grammars.

## Exercises

Mastering the theoretical foundations of computer science is not a spectator sport; only by solving problems and examining the proofs of the major results can one fully comprehend the concepts, the algorithms, and the subtleties of the theory. That is, understanding the "big picture" requires many small steps. To help accomplish this, each chapter ends with a set of exercises. The exercises range from constructing simple examples of the topics introduced in the chapter to extending the theory.

Several exercises in each set are marked with a star. A problem is starred because it may be more challenging than the others on the same topic, more theoretical in nature, or may be particularly unique and interesting.

## Notation

The theory of computer science is a mathematical examination of the capabilities and limitations of effective computation. As with any formal analysis, the notation must provide precise and unambiguous definitions of the concepts, structures, and operations. The following notational conventions will be used throughout the book:

| Items | Description | Examples |
|---|---|---|
| Elements and strings | Italic lowercase letters from the beginning of the alphabet | $a, b, abc$ |
| Functions | Italic lowercase letters | $f, g, h$ |
| Sets and relations | Capital letters | X, Y, Z, Σ, Γ |
| Grammars | Capital letters | G, $\mathrm{G}_1$, $\mathrm{G}_2$ |
| Variables of grammars | Italic capital letters | $A, B, C, S$ |
| Abstract machines | Capital letters | M, $\mathrm{M}_1$, $\mathrm{M}_2$ |

The use of roman letters for sets and mathematical structures is somewhat nonstandard but was chosen to make the components of a structure visually identifiable. For example, a context-free grammar is a structure $\mathrm{G} = (\Sigma, \mathrm{V}, \mathrm{P}, S)$. From the fonts alone it can be seen that G consists of three sets and a variable $S$.

A three-part numbering system is used throughout the book; a reference is given by chapter, section, and item. One numbering sequence records definitions, lemmas, theorems, corollaries, and algorithms. A second sequence is used to identify examples. Tables, figures, and exercises are referenced simply by chapter and number.

The end of a proof is marked by ■ and the end of an example by □. An index of symbols, including descriptions and the numbers of the pages on which they are introduced, is given in Appendix I.

## Supplements

Solutions to selected exercises are available only to qualified instructors. Please contact your local Addison-Wesley sales representative or send email to *aw.cse@aw.com* for information on how to access them.

## Acknowledgments

First and foremost, I would like to thank my wife Janice and daughter Elizabeth, whose kindness, patience, and consideration made the successful completion of this book possible. I would also like to thank my colleagues and friends at the Institut de Recherche en Informatique de Toulouse, Université Paul Sabatier, Toulouse, France. The first draft of this revision was completed while I was visiting IRIT during the summer of 2004. A special thanks to Didier Dubois and Henri Prade for their generosity and hospitality.

The number of people who have made contributions to this book increases with each edition. I extend my sincere appreciation to all the students and professors who have

used this book and have sent me critiques, criticisms, corrections, and suggestions for improvement. Many of the suggestions have been incorporated into this edition. Thank you for taking the time to send your comments and please continue to do so. My email address is *tsudkamp@cs.wright.edu.*

This book, in its various editions, has been reviewed by a number of distinguished computer scientists including Professors Andrew Astromoff (San Francisco State University), Dan Cooke (University of Texas-El Paso), Thomas Fernandez, Sandeep Gupta (Arizona State University), Raymond Gumb (University of Massachusetts-Lowell), Thomas F. Hain (University of South Alabama), Michael Harrison (University of California at Berkeley), David Hemmendinger (Union College), Steve Homer (Boston University), Dan Jurca (California State University-Hayward), Klaus Kaiser (University of Houston), C. Kim (University of Oklahoma), D. T. Lee (Northwestern University), Karen Lemone (Worcester Polytechnic Institute), C. L. Liu (University of Illinois at Urbana-Champaign), Richard J. Lorentz (California State University-Northridge), Fletcher R. Norris (The University of North Carolina at Wilmington), Jeffery Shallit (University of Waterloo), Frank Stomp (Wayne State University), William Ward (University of South Alabama), Dan Ventura (Brigham Young University), Charles Wallace (Michigan Technological University), Kenneth Williams (Western Michigan University), and Hsu-Chun Yen (Iowa State University). Thank you all.

I would also like to gratefully acknowledge the assistance received from the people at the Computer Science Education Division of the Addison-Wesley Publishing Company and Windfall Software who were members of the team that successfully completed this project.

Thomas A. Sudkamp
*Dayton, Ohio*

# Contents

## PART II Grammars, Automata, and Languages