Preface

This book is designed to serve as a text for a one-semester introductory course in the theory of formal languages and computation. It covers all the traditional topics of this theory, such as automata, grammars, parsing, computability, decidability, computational complexity, and properties of formal languages. Special attention is paid to the fundamental models for formal languages and their applications in computer science.

Subject

Formal language theory defines languages mathematically as sets of sequences consisting of symbols. This definition encompasses almost all languages as they are commonly understood. Indeed, natural languages, such as English, are included in this definition. Of course, all artificial languages introduced by various scientific disciplines can be viewed as formal languages; perhaps most illustratively, every programming language represents a formal language in terms of this definition. It thus comes as no surprise that formal language theory, which represents a mathematically systematized body of knowledge concerning formal languages, is important to all the scientific areas that make use of these languages to a certain extent.

The theory of formal languages represents the principal subject of this book. The text focuses its attention on the fundamental models for formal languages and their computation-related applications in computer science, hence its title *Formal Languages and Computation*: *Models and Their Applications*.

Models

The strictly mathematical approach to languages necessitates introducing formal models that define them. Most models of this kind are underlain by *rewriting systems*, which are based on rules by which they repeatedly change sequences of symbols, called strings. Despite their broad variety, most of them can be classified into two basic categories—generative and accepting language models. Generative models, better known as *grammars*, define strings of their language, so their rewriting process generates them from a special start symbol. On the other hand, accepting models, better known as *automata*, define strings of their language by a rewriting process that starts from these strings and ends in a prescribed set of final strings.

Applications

The book presents applications of language models in both practical and theoretical computer science.

In practice, the text explains how appropriate language models underlie computer science engineering techniques used in *language processors*. It pays special attention to *programming language analyzers* based on four language models—regular expressions, finite automata, context-free grammars, and pushdown automata. More specifically, by using *regular expressions* and *finite automata*, it builds up *lexical analyzers*, which recognize lexical units and verify that they are properly formed. Based on *context-free grammars* and *pushdown automata*, it creates *syntax analyzers*, which recognize syntactic structures in computer programs and verify that they are correctly written according to grammatical rules. That is, the text first explains how to specify the programming language syntax by using context-free grammars, which are considered the most widely used specification tool for this purpose. Then, it describes how to write syntax analyzers based on pushdown automata.

In theory, the book makes use of language-defining models to explore the very heart of the *foundations of computation*. That is, the text introduces the mathematical notion of a *Turing machine*, which has become a universally accepted formalization of the intuitive notion of a procedure. Based on this strictly mathematical notion, it studies the general limits of computation. More specifically, it performs this study in terms of two important topics concerning computation—computability and decidability. Regarding *computability*, it considers Turing machines as computers of functions over nonnegative integers and demonstrates the existence of functions whose computation cannot be specified by any procedure. As far as *decidability* is concerned, it formalizes problem-deciding algorithms by Turing machines that halt on every input. The book formulates several important problems concerning the language models discussed earlier in this book and constructs algorithms that decide them. On the other hand, it describes several problems that are not decidable by any algorithm. Apart from giving several specific undecidable problems, this book builds up a general theory of undecidability. Finally, the text approaches decidability in a much finer and realistic way. Indeed, it reconsiders problem-deciding algorithms in terms of their computational complexity measured according to time and space requirements. Perhaps most importantly, it shows that although some problems are decidable in principle, they are intractable for absurdly high computational requirements of the algorithms that decide them.

Use

As already stated, this book is intended as a textbook for a one-term introductory course in formal language theory and its applications in computer science.

Second, the book can also be used as an accompanying textbook for a compiler class at an undergraduate level because the text allows the flexibility needed to select only the topics relevant to compilers.

Finally, this book is useful to all researchers, including people out of computer science, who somehow deal with formal languages and their models in their scientific fields.

Approach

Primarily, this book represents a theoretically oriented treatment of formal languages and their models. Indeed, it introduces all formalisms concerning them with enough rigor to make all results quite clear and valid. Every complicated mathematical passage is preceded

by its intuitive explanation so that even the most complex parts of the book are easy to grasp. Every new concept or algorithm is preceded by an explanation of its purpose and followed by some examples with comments to reinforce its understanding. All applications are given in a quite realistic way to clearly demonstrate a strong relation between the theoretical concepts and their uses.

Secondarily, as already pointed out, the text also presents several significant applications of formal languages and their models in practice. All applications are given in a quite realistic way to clearly show a close relation between the theoretical concepts and their uses.

Prerequisites

On the part of the student, no previous knowledge concerning formal languages is assumed. Although this book is self-contained, in the sense that no other sources are needed for understanding the material, a familiarity with the rudiments of discrete mathematics is helpful for a quick comprehension of formal language theory. A familiarity with a high-level programming language helps to grasp the material concerning applications in this book.

Organization

Synopsis

The entire text contains 12 chapters, which are divided into 5 sections.

Section I, which consists of Chapters 1 and 2, gives an introduction to the subject. Chapter 1 recalls the basic mathematical notions used in the book. Chapter 2 gives the basics of formal languages and rewriting systems that define them.

Section II, which consists of Chapters 3 through 5, studies regular languages and their models. Chapter 3 gives the basic definitions of these languages and their models, such as regular expressions and finite automata. Chapter 4 is application oriented; specifically, it builds lexical analyzers by using models for regular languages. Chapter 5 studies properties concerning regular languages.

Section III, which consists of Chapters 6 through 8, discusses context-free languages and their models. To a large extent, its structure parallels Section II. Indeed, Chapter 6 defines context-free languages and their models, including context-free grammars and pushdown automata. Chapter 7 explains how to construct syntax analyzers based on these grammars and automata. Chapter 8 establishes certain properties concerning context-free languages.

Section IV, which consists of Chapters 9 through 11, concerns Turing machines as a formalization of algorithms. Chapter 9 defines them. Based on Turing machines, Chapter 10 gives the basic ideas, concepts, and results underlying the theory of computation and its crucially important parts, including computability, decidability, and computational complexity. Simultaneously, this chapter establishes important properties concerning languages defined by Turing machines. Chapter 11 presents the essentials concerning general grammars, which represent grammatical counterparts to Turing machines.

Section V consists of Chapter 12. This chapter summarizes the entire textbook, points out selected modern trends, makes many historical and bibliographical remarks, and recommends further reading to the serious student.

Finally, the book contains two appendices. Appendix I gives the index to mathematical symbols used in the text. Appendix II contains the alphabetic index that lists all important language models introduced in the book.

Numbering

Regarding the technical organization of the text, algorithms, conventions, corollaries, definitions, lemmas, and theorems are sequentially numbered within chapters. Examples and figures are organized similarly. The end of conventions, corollaries, definitions, lemmas, and theorems is denoted by ■.

Exercises

At the end of each chapter, a set of exercises is given to reinforce and augment the material covered. Selected exercises, denoted by S, have their solutions or parts of them at the end of the chapter.

Algorithms

This textbook contains many algorithms. Strictly speaking, every algorithm requires a verification that it terminates and works correctly. However, the termination of the algorithms given in this book is always so obvious that its verification is omitted throughout. The correctness of complicated algorithms is verified in detail. On the other hand, we most often give only the gist of the straightforward algorithms and leave their rigorous verification as an exercise. The text describes the algorithms in Pascal-like notation, which is so simple and intuitive that even the student unfamiliar with the Pascal programming language can immediately pick it up. In this description, a Pascal **repeat** loop is sometimes ended with **until no change**, meaning that the loop is repeated until no change can result from its further repetition. As the clear comprehensibility is a paramount importance in the book, the description of algorithms is often enriched by an explanation in words.

Support on the World Wide Web

Further backup materials, including lecture notes, are available at http://www.fit.vutbr.cz/~meduna/books/flc.

Acknowledgments

For almost a decade, I taught the theory of formal languages and computation at the University of Missouri-Columbia in the United States back in the 1990s, and since 2000, I have taught this subject at the Brno University of Technology in the Czech Republic. The lecture notes I wrote at these two universities underlie this book, and I have greatly benefited from conversations with many colleagues and students there. In addition, this book is based on notes I have used for my talks at various American, Asian, and European universities over the past three decades. Notes made at the Kyoto Sangyo University in Japan were particularly helpful.

Writing this book was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070). This work was also supported by the Visual Computing Competence Center (TE01010415).

My thanks to Martin Čermák and Jiří Techet for their comments on a draft of this text. Without the great collaboration, encouragement, and friendship with Zbyněk Křivka, I would have hardly started writing this book, let alone complete it. I am also grateful to John Wyzalek at Taylor & Francis for excellent editorial work. Finally, I thank my wife Ivana for her patience and, most importantly, love.

Author

Alexander Meduna, PhD, is a full professor of computer science at the Brno University of Technology in the Czech Republic, where he earned his doctorate in 1988. From 1988 until 1997, he taught computer science at the University of Missouri-Columbia in the United States. Even more intensively, since 2000, he has taught computer science and mathematics at the Brno University of Technology. In addition to these two universities, he has taught computer science at several other American, European, and Japanese universities for shorter periods of time. His classes have been primarily focused on formal language theory and its applications in theoretical and practical computer science. His teaching has also covered various topics including automata, discrete mathematics, operating systems, and principles of programming languages. Among many other awards for his scholarship and writing, he received the Distinguished University Professor Award from Siemens in 2012. He very much enjoys teaching classes related to the subject of this book.

Dr. Meduna has written several books. Specifically, he is the author of two textbooks—*Automata and Languages* (Springer, 2000) and *Elements of Compiler Design* (Taylor & Francis, 2008; translated into Chinese in 2009). Furthermore, he is the coauthor of three monographs—*Grammars with Context Conditions and Their Applications* (along with Martin Švec, Wiley, 2005), *Scattered Context Grammars and Their Applications* (with Jiří Techet, WIT Press, 2010), and *Regulated Grammars and Automata* (with Petr Zemek, Springer, 2014). He has published over 90 studies in prominent international journals, such as *Acta Informatica* (Springer), *International Journal of Computer Mathematics* (Taylor & Francis), and *Theoretical Computer Science* (Elsevier). All his scientific work discusses the theory of formal languages and computation, the subject of this book, or closely related topics, such as compiler writing.

Alexander Meduna's website is http://www.fit.vutbr.cz/~meduna. His scientific work is described in detail at http://www.fit.vutbr.cz/~meduna/work.