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 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 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.

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. The more difficult exercises are marked with a star. 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.

The fifth edition of this text introduces a substantial amount of new material. While the presentation in the fourth edition has been retained with only minor modifications, two appendices have been added. The first is an entire chapter on finite-state transducers, Appendix A. While transducers play no significant role in formal language theory, they are important in other areas of computer science, such as digital design. Students can benefit from an early exposure to this subject; if time permits it is worthwhile to do so. Due to the similarity with finite accepters, this involves few new concepts.

I also added an introduction to JFLAP, an interactive software tool that I feel is of great help in both learning the material and in teaching this course. JFLAP implements most of the ideas and constructions in this book. It not only helps students visualize abstract concepts, but it is also a great time-saver. Many of the exercises in this book require creating structures that are complicated and that have to be thoroughly tested for correctness. JFLAP can reduce the time required for this by an order of magnitude. Appendix B gives a brief introduction to JFLAP and the CD that comes with the book expands on this. I very much recommend the use of JFLAP for both students and instructors.

Peter Linz