This is page
Printer: Opaque this

Computability and Complexity Theory[1]

Steven Homer
Department of Computer Science
Boston University
Boston, MA 02215

Alan L. Selman
Department of Computer Science and Engineering
University at Buffalo
226 Bell Hall
Buffalo, NY 14260

August 24, 2000

We dedicate this book to our wives, Michelle and Sharon.

Preface

The theory of computing provides computer science with concepts, models, and formalisms for reasoning about the resources needed to carry out computations and about the efficiency of the computations that use these resources. It provides tools to measure the difficulty of combinatorial problems both absolutely and in comparison with other problems. Courses in this subject help students to gain analytic skills and enable them to recognize the limits of computation. For these reasons, a course in theory of computing is usually required in the graduate computer science curriculum.

The harder question to address is which topics such a course should cover. We believe that students should learn the fundamental models of computation, the limitations of computation, and the distinctions between feasible and intractable. In particular, the phenomena of NP-completeness and NP-hardness have pervaded much of science and transformed computer science. One option is to survey a large number of theoretical subjects, typically focusing on automata and formal languages. However, these subjects are less important to theoretical computer science, and to computer science as a whole, now than in the past. Many students have taken such a course as part of their undergraduate education. We chose not to take that route because computability and complexity theory are the subjects that we feel deeply about and that we believe are important for students to learn. Furthermore, a graduate course should be scholarly. It is better to treat important topics thoroughly than to survey the field.

This textbook is intended for use in an introductory graduate course in theoretical computer science. It contains material that should be core knowledge in the theory of computation for all graduate students in computer science. It is self-contained and is best suited for a one semester course. Most of the text can be covered in one semester by moving expeditiously through the core material of Chapters 1 through 5 and then covering parts of Chapter 6. We will give more details about this below.

As a graduate course, students should have some prerequisite preparation. The ideal preparation would be the kind of course that we mentioned above: an undergraduate course that introduced topics such as automata theory, formal languages, computability theory, or complexity theory. We stress, however, that there is nothing in such a course that a student needs to know before studying this text. Our personal experience suggests that we cannot presume that all of our students have taken such an undergraduate course. For those students who have not, we advise that they need at least some prior exposure that will have developed mathematical skills. Prior courses in mathematical logic, algebra (at the level of groups, rings, or fields), or number theory, for example, would all serve this purpose.

Despite the diverse backgrounds of our students, we have found that graduate students are capable of learning sophisticated material when it is explained clearly and precisely. That has been our goal in writing this book.

This book also is suitable for advanced undergraduate students who have satisfied the prerequisites. It is an appropriate first course in complexity theory for students who will continue to study and work in this subject area.

The text begins with a preliminary chapter that gives a brief description of several topics in mathematics. We included this in order to keep the book self-contained and to insure that all students will have a common notation. Some of these sections simply enable students to understand some of the important examples that arise later. For example, we include a section on number theory and algebra that includes all that is necessary for students to understand that primality belongs to NP.

The text starts properly with classical computability theory. We build complexity theory on top of that. Doing so has the pedagogical advantage that students learn a qualitative subject before advancing to a quantitative one. Also, the concepts build from one to the other. For example, although we give a complete proof that the satisfiability problem is NP-complete, it is easy for students to understand that the bounded halting problem is NP-complete, because they already know that classical halting problem is c.e. complete.

We use the terms *partial computable* and *computably enumerable (c.e.)* instead of the traditional terminology, *recursive* and *recursively enumerable (r.e.)*, respectively. We do so simply to eliminate confusion. Students of computer science know of "recursion" as a programming paradigm. We do not prove here that Turing computable functions are equivalent to partial recursive functions, so by not using that notation, we avoid the matter altogether. Although the notation we are using has been commonplace in the computability theory and mathematical logic community for several years, instructors might want to advise their students that the older terminology seems commonplace within the theoretical computer science community. Computable functions are defined on the set of words over a finite alphabet, which we identify with the set of natural numbers in a straightforward manner. We use the term *effective*, in the nontechnical, intuitive sense, to denote computational processes on other data types. For example, we will say that a set of Turing machines is "effectively enumerable" if its set of indices is computably enumerable.

Chapter 3 concludes with a short list of topics that students should know from the chapters on computability theory before proceeding to study complexity theory. We advise instructors who wish to minimize coverage of computability theory to refer to this list. Typically, we do not cover the second section on the Recursion Theorem (Section 2.10) in a one-semester course. Although we do not recommend it, it is possible to begin the study of complexity theory after learning the first five sections of Chapter 3 and at least part of Section 2.9 on oracle Turing machines, Turing reductions, and the arithmetical hierarchy.

In Chapter 4 we treat general properties of complexity classes and relationships between complexity classes. These include important older results such as the space and time hierarchy theorems, as well as the more recent result of Immerman and Szelepcsényi that space-bounded classes are closed under complements. Instructors might be anxious to get to NP-complete problems, Chapter 5, and NP-hard problems, Chapter 6, but students need to learn the basic results of complexity theory and it is instructive for them to understand the relationships between P, NP, and other deterministic and nondeterministic, low-level complexity classes. Students should learn that nondeterminism is not well understood in general, that P =? NP is not an isolated question, and that other classes have complete problems as well (which we take up in Chapter 6). Nevertheless, Chapter 4 is a long chapter. Many of the results in this chapter are proved by complicated Turing machine simulations and counting arguments, which give students great insight, but can be time-consuming to cover. For this reason, instructors might be advised to survey some of this material, if the alternative would mean not having sufficient time for the later chapters.

Homework exercises are an important part of this book. They are embedded in the text where they naturally arise and students should not proceed without working on them. Many are simple exercises while others are challenging. Often we leave important but easy-to-prove propositions as exercises. We provide additional problems at the end of chapters, which extend and apply the material covered there.

Once again, our intent has been to write a text that is suitable for all graduate students, that provides the right background for those who will continue to study complexity theory, and that can be taught in one semester. There are several important topics in complexity theory that cannot be treated properly in a one-semester course. Currently we are writing a second part to this text, which would be suitable for an optional second semester course, covering nonuniform complexity (Boolean circuits), parallelism, probabilistic classes, and interactive protocols.

S. Homer — Boston
A. Selman — Buffalo

Contents

Preface **vi**

0 Preliminaries **1**
- 0.1 Words and Languages . . . 1
- 0.2 K-adic representation . . . 2
- 0.3 Partial Functions . . . 3
- 0.4 Graphs . . . 4
- 0.5 Propositional Logic . . . 6
  - 0.5.1 Boolean functions . . . 8
- 0.6 Cardinality . . . 8
  - 0.6.1 Ordered Sets . . . 11
- 0.7 Elementary Algebra . . . 11
  - 0.7.1 Rings and Fields . . . 12
  - 0.7.2 Groups . . . 16
  - 0.7.3 Number Theory . . . 18

1 Introduction to Computability **24**
- 1.1 Turing machines . . . 25
- 1.2 Turing machine concepts . . . 27
- 1.3 Variations of Turing machines . . . 30
  - 1.3.1 Multitape Turing Machines . . . 30
  - 1.3.2 Nondeterministic Turing Machines . . . 33
- 1.4 Church's thesis . . . 36
- 1.5 RAMs . . . 37

1.5.1 Turing machines for RAMS 41

2 Undecidability 42

2.1 Decision Problems 42
2.2 Undecidable Problems 44
2.3 Pairing Functions 46
2.4 Computably Enumerable Sets 47
2.5 Halting Problem, Reductions, and Complete Sets 52
2.5.1 Complete Problems 54
2.6 *S-m-n* Theorem 54
2.7 Recursion Theorem 57
2.8 Rice's Theorem 60
2.9 Turing Reductions and Oracle Turing Machines 61
2.10 Recursion Theorem, Continued 68
2.11 References 72

3 Introduction to Complexity Theory 75

3.1 Complexity Classes and Complexity Measures 77
3.1.1 Computing Functions 79
3.2 Prerequisites 80

4 Basic Results of Complexity Theory 81

4.1 Linear Compression and Speedup 82
4.2 Constructible Functions 89
4.2.1 Simultaneous Simulation 90
4.3 Tape Reduction 93
4.4 Inclusion Relationships 99
4.4.1 Relations between the Standard Classes 108
4.5 Separation Results 110
4.6 Translation Techniques and Padding 114
4.6.1 Tally Languages 116
4.7 Relations between the Standard Classes—Continued 117
4.7.1 Complements of Complexity Classes: The Immerman–Szelepcsényi Theorem 119

5 Nondeterminism and NP-Completeness 125

5.1 Characterizing NP 126
5.2 The Class P 127
5.3 Enumerations 129
5.4 NP-completeness 131
5.5 The Cook–Levin Theorem 133
5.6 More NP-complete problems 139
5.6.1 The diagonal set is NP-complete 139
5.6.2 Some natural NP-complete problems 140

6 Relative Computability 148

6.1 NP-hardness 150
6.2 Search Problems 154
6.3 The Structure of NP 156
6.3.1 Composite Number and Graph Isomorphism 162
6.3.2 Reflection 165
6.4 The Polynomial Hierarchy 165
6.5 Complete Problems for Other Complexity Classes 174
6.5.1 PSPACE 174
6.5.2 Exponential Time 178
6.5.3 Polynomial Time and Logarithmic Space 179
6.5.4 A Note on Provably Intractable Problems 183

References **185**