# Preface

## *Context*

The paradoxes discovered in Cantor's set theory sometime around 1900 began a crisis that shook the foundations of mathematics. In order to reconstruct mathematics, freed from all paradoxes, Hilbert introduced a promising program with formal systems as the central idea. Though the program was unexpectedly brought to a close in 1931 by Gödel's famous theorems, it bequeathed burning questions: "What is computing? What is computable? What is an algorithm? Can every problem be algorithmically solved?" This led to *Computability Theory*, which was born in the mid-1930s, when these questions were resolved by the seminal works of Church, Gödel, Kleene, Post, and Turing. In addition to contributing to some of the greatest advances of twentieth century mathematics, their ideas laid the foundations for the practical development of a universal computer in the 1940s as well as the discovery of a number of algorithmically unsolvable problems in different areas of science. New questions, such as "Are unsolvable problems equally difficult? If not, how can we compare their difficulty?" initiated new research topics of *Computability Theory*, which in turn delivered many important concepts and theorems. The application of these is central to the multidisciplinary research of *Computability Theory*.

## *Aims*

Monographs in *Theoretical Computer Science* usually strive to present as much of the subject as possible. To achieve this, they present the subject in a definition–theorem–proof style and, when appropriate, merge and intertwine different related themes, such as computability, computational complexity, automata theory, and formal language theory. This approach, however, often blurs historical circumstances, reasons, and the motivation that led to important goals, concepts, methods, and theorems of the subject.

My aim is to compensate for this. Since the fundamental ideas of theoretical computer science were either motivated by historical circumstances in the field, or were developed by pure logical reasoning, I describe *Computability Theory*, a part of *Theoretical Computer Science*, from this point of view. Specifically, I describe difficulties that arose in mathematical logic, the attempts to recover from them, and how these attempts led to the birth of *Computability Theory* and later influenced it. Although some of these attempts fell short of their primary goals, they put forward crucial questions about computation and led to the fundamental concepts of *Computability Theory*. These in turn logically led to still new questions, and so on. By describing this evolution I want to give the reader a deeper understanding of the foundations of this beautiful theory. The challenge in writing this book was therefore to keep it accessible by describing the historical and logical development while at the same time introducing as many modern topics as needed to start the research. Thus, I will be happy if the book makes good reading before one tackles more advanced literature on *Computability Theory*.

## *Contents*

There are three parts in this book.

**Part I** (Chaps. 1–4) ***Chapter 1*** is introductory: it discusses the *intuitive* comprehension of the concept of the algorithm. This comprehension was already provided by Euclid and sufficed since 300 B.C.E. or so. In the next three chapters I explain how the need for a rigorous, *mathematical definition* of the concepts of the algorithm, computation, and computability was born. ***Chapter 2*** describes the events taking place in mathematics around 1900, when *paradoxes* were discovered. The circumstances that led to the paradoxes and consequently to the foundational crisis in mathematics are explained. The ideas of the three main schools of recovery—the *intuitionism*, *logicism*, and *formalism*—that attempted to reconstruct mathematics, are described. ***Chapter 3*** delves into formalism. This school gathered the ideas and results of other schools in the concept of the *formal axiomatic system*. Three particular systems that played crucial roles in the future events are described; these are the formal axiomatic systems of *logic*, *arithmetic*, and *set theory*. ***Chapter 4*** presents *Hilbert's Program*, a promising formalistic attempt that would use formal axiomatic systems to eliminate all the paradoxes from mathematics. It is explained how *Hilbert's Program* was unexpectedly shattered by Gödel's *Incompleteness Theorems*, which state, in effect, that not every truth can be proved (in a formal system).

**Part II** (Chaps. 5–9) *Hilbert's Program* left open a question about the existence of a particular algorithm, the algorithm that would solve the *Entscheidungsproblem*. Since this algorithm might not exist, it was necessary to formalize the concept of the algorithm—only then would a proof of the non-existence of the algorithm be possible. Therefore, ***Chapter 5*** discusses the fundamental questions: "What is an algorithm? What is computation? What does it mean when we say that a function or problem is computable?" It is explained how these intuitive, informal concepts were formally defined in the form of the *Computability* (*Church–Turing*) *Thesis* by

the different yet equivalent *models of computation*, such as partial and general recursive functions, $\lambda$-calculus, the Turing machine, the Post machine, and the Markov algorithms. ***Chapter 6*** focuses on the *Turing machine* that most convincingly formalized the intuitive concepts of computation. Several equivalent variants of the Turing machine are described. Three basic uses of the Turing machines are presented: function computation, set generation, and set recognition. The existence of the *universal Turing machine* is proved and its impact on the creation and development of *general-purpose computers* is described. The equivalence of the Turing machine and the RAM model of computation is proved. In ***Chapter 7***, the first basic yet important theorems are deduced. These include the relations between *decidable*, *semi-decidable*, and *undecidable sets* (i.e., decision problems), the *Padding Lemma*, the *Parametrization* (i.e., *s-m-n*) *Theorem*, and the *Recursion Theorem*. The latter two are also discussed in view of the recursive procedure calls in the modern general-purpose computer. ***Chapter 8*** is devoted to *incomputability*. It uncovers a surprising fact that, in effect, not everything that is defined can be computed (on a usual model of computation). Specifically, the chapter shows that not every computational problem can be solved by a computer. First, the incomputability of the *Halting Problem* is proved. To show that this is not just a unique event, a list of selected incomputable problems from various fields of science is given. Then, in ***Chapter 9***, methods of proving the incomputability of problems are explained; in particular, proving methods by *diagonalization*, *reduction*, the *Recursion Theorem*, and *Rice's Theorem* are explained.

**Part III** (Chaps. 10–15) In this part attention turns to *relative computability*. I tried to keep the chapters "bite-sized" by focusing in each on a single issue only. ***Chapter 10*** introduces the concepts of the oracle and the *oracle Turing machine*, describes how computation with such an external help would run, and discusses how oracles could be replaced in the real world by actual databases or networks of computers. ***Chapter 11*** formalizes the intuitive notion of the "*degree of unsolvability*" of a problem. To do this, it first introduces the concept of *Turing reduction*, the most general reduction between computational problems, and then the concept of *Turing degree*, which formalizes the notion of the degree of unsolvability. This formalization makes it possible to define, in ***Chapter 12***, an operator called *Turing jump* and, by applying it, to construct a *hierarchy* of infinitely many Turing degrees. Thus, a surprising fact is discovered that for every unsolvable problem there is a more difficult unsolvable problem; there is no most difficult unsolvable problem. ***Chapter 13*** embarks on this intriguing fact. It first introduces a view of the *class of all Turing degrees* as a mathematical structure. This eases expression of relations between the degrees. Then several properties of this class are proved, revealing a highly complex structure of the class. ***Chapter 14*** introduces *computably enumerable* (*c.e.*) *Turing degrees*. It then presents *Post's Problem*, posing whether there exist c.e. degrees other than $\mathbf{0}$ below the degree $\mathbf{0}'$. Then the *priority method*, discovered and used by Friedberg and Muchnik to solve *Post's Problem*, is described. ***Chapter 15*** introduces the *arithmetical hierarchy*, which gives another, arithmetical view of the degrees of unsolvability. Finally, ***Chapter 16*** lists some suggestions for further reading.

## *Approach*

The main lines of the approach are:

- ***Presentation levels.*** I use two levels of presentation, the fast track and detours. The *fast track* is a *fil rouge* through the book and gives a bird's-eye view of *Computability Theory*. It can be read independently of *detours*. These contain detailed proofs, more demanding themes, additional historical facts, and further details, all of which can safely be skipped while reading on the fast track. The two levels differ visually: detours are written in small font and are put into *Boxes* (between gray bars, with broken lower bar), so they can easily be skipped or skimmed on first reading. Proofs are given on both levels whenever they are difficult or long.

- ***Clarity.*** Whenever possible I give the motivation and an explanation of the circumstances that led to new goals, concepts, methods, or theorems. For example, I explicitly point out with **NB** (nota bene) marks those situations and achievements that had important impact on further development in the field. Sometimes **NB** marks introduce conventions that are used in the rest of the book. New notions are introduced when they are naturally needed. Although I rigorously deduce theorems, I try to make proofs as intelligible as possible; this I do by commenting on tricky inferences and avoiding excessive formalism. I give intuitive, informal explanations of the concepts, methods, and theorems. Figures are given whenever this can add to the clarity of the text.

- ***Contemporary terminology.*** I use the recently suggested terminology and describe the reasons for it in the Bibliographic Notes; thus, I use partial computable (p.c.) functions (instead of partial recursive (p.r.) functions); computable functions (instead of recursive functions); computably enumerable (c.e.) sets (instead of recursively enumerable (r.e.) sets); and computable sets (instead of recursive sets).

- ***Historical account.*** I give an extended historical account of the mathematical and logical roots of *Computability Theory*.

- ***Turing machine.*** After describing different competing models of computation, I adopt the Turing machine as *the* model of computation and build on it. I neither formally prove the equivalence of these models, nor do I teach how to program Turing machines; I believe that all of this would take excessive space and add little to the understanding of *Computability Theory*. I do, however, rigorously prove the equivalence of the Turing machine and the RAM model, as the latter so closely abstracts real-life, general-purpose computers.

- ***Unrestricted computing resources.*** I decouple *Automata Theory* and *Formal Language Theory* from *Computability Theory*. This enables me to consider general models of computation (i.e., models with unlimited resources) and hence

focus freely on the question "What can be computed?" In this way, I believe, *Computability Theory* can be seen more clearly and it can serve as a natural basis for the development of *Computational Complexity Theory* in its study of "What can be computed efficiently?" Although I don't delve into *Computational Complexity Theory*, I do indicate the points where *Computational Complexity Theory* would take over.

- ***Short-cuts to relative computability.*** I introduce oracles in the usual way, after explaining classical computability. Readers eager to enter relative computability might want to start with Part II and continue on the fast track.

- ***Practical consequences and applications.*** I describe the applications of concepts and theorems, whenever I am aware of them.

Finally, in describing *Computability Theory* I do not try to be comprehensive. Rather, I view the book as a first step towards more advanced texts on *Computability Theory*, or as an introductory text to *Computational Complexity Theory*.

## *Audience*

This book is written at a level appropriate for undergraduate or beginning graduate students in computer science or mathematics. It can also be used by anyone pursuing research at the intersection of theoretical computer science on the one hand and physics, biology, linguistics, or analytic philosophy on the other.

The only necessary prerequisite is some exposure to elementary logic. However, it would be helpful if the reader has had undergraduate-level courses in set theory and introductory modern algebra. All that is needed for the book is presented in App. A, which the reader can use to fill in the gaps in his or her knowledge.

## *Teaching*

There are several courses one can teach from this book. A course offering the *minimum* of *Computability Theory* might cover (omitting boxes) Chaps. 5, 6, 7; Sects. 8.1, 8.2, 8.4; and Chap. 9. Such a course might be continued with a course on *Complexity Theory*. An *introductory* course on *Computability Theory* might cover Parts I and II (omitting most boxes of Part I). A beginning *graduate* level course on *Computability Theory* might cover all three parts (with all the details in boxes). A course offering a *shortcut* (some 60 pages) to *Relative Computability* (Chaps. 10 to 15) might cover Sect. 5.3; Sects. 6.1.1, 6.2.1, 6.2.2; Sects. 6.3, 7.1, 7.2, 7.3; Sects. 7.4.1, 7.4.2, 7.4.3; Sects. 8.1, 8.2, 9.1, 9.2; and then Chaps. 10 through 15.

PowerPoint slides covering all three parts of the text are maintained and available at:

`http://lalg.fri.uni-lj.si/fct`

## *Origin*

This book grew out of two activities: (1) the courses in *Computability and Computational Complexity Theory* that I have been teaching at the University of Ljubljana, and (2) my intention to write a textbook for a course on algorithms that I also teach.

When I started working on (2) I wanted to explain the $\mathcal{O}$-notation in a satisfactory way, so I planned an introductory chapter that would cover the basics of *Computational Complexity Theory*. But to explain the latter in a satisfactory way, the basics of *Computability Theory* had to be given first. So, I started writing on computability. But the story repeated once again and I found myself describing the *Mathematical Logic* of the twentieth century. This regression was due to (i) my awareness that, in the development of mathematical sciences, there was always some *reason* for introducing a new notion, concept, method, or goal, and (ii) my belief that the text should describe such reasons in order to present the subject as clearly as possible. Of course, many historical events and logical facts were important in this respect, so the chapter on *Computability Theory* continued to grow.

At the same time, I was aware that students of *Computability and Computational Complexity Theory* often have difficulty in grasping the meaning and importance of certain themes, as well as in linking up the concepts and theorems to a whole. It was obvious that before introducing a new concept, method, or goal, a student should be given a historical or purely logical *motivation* for such a step. In addition, giving a *bird's eye view* of the theory developed up to the last milestone also proved to be extremely advantageous.

These observations coincided with my wishes about the chapter on *Computability Theory*. So the project continued in this direction until the "chapter" grew into a text on *The Foundations of Computability Theory*, which is in front of you.

## *Acknowledgements*

I would like to express my sincere thanks to all the people who read all or parts of the manuscript and suggested improvements, or helped me in any other way. I benefited from the comments of my colleagues *Uroš Čibej* and *Jurij Mihelič*. In particular, *Marko Petkovšek* (University of Ljubljana, Faculty of Mathematics and Physics, Department of Mathematics), and *Danilo Šuster* (University of Maribor, Faculty of Arts, Department of Philosophy) meticulously read the manuscript and suggested many improvements. The text has benefited enormously from their assistance. Although errors may remain, these are entirely my responsibility.

Many thanks go to my colleague *Boštjan Slivnik* who skilfully helped me on several occasions in getting over TEX and its fonts. I have used drafts of this text in courses on *Computability and Computational Complexity Theory* that are given to students of computer science by our faculty, and to students of computer science and mathematics in courses organized in collaboration with the Faculty of Mathematics and Physics, University of Ljubljana. For their comments I particularly thank the students *Žiga Emeršič, Urša Krevs, Danijel Mišanović, Rok Resnik, Blaž Sovdat, Tadej Vodopivec,* and *Marko Živec.* For helpful linguistic suggestions, discussions on the pitfalls of English, and careful proofreading I thank *Paul McGuiness.*

I have made every reasonable effort to get permissions for including photos of the scientists whose contributions to the development of *Computability Theory* are described in the book. It turned out that most of the photos are already in the public domain. Here, *Wikimedia* makes praiseworthy efforts in collecting them; so does the online *MacTutor History of Mathematics Archive* at the University of St Andrews, Scotland. They were both very helpful and I am thankful to them. For the other photos I owe substantial thanks to the *Archives of the Mathematisches Forschungsinstitut Oberwolfach*, Germany, *King's College Library*, Cambridge, UK, and the *Los Alamos National Laboratory Archives*, USA. I have no doubt that photos make this serious text more pleasant. The following figures are courtesy of Wikimedia: Figs. 1.3, 1.4, 1.5, 1.7, 2.5, 2.6, 2.7, 2.9, 3.5, 3.8, 5.4, and 5.10. The following figures are courtesy of the MacTutor History of Mathematics archive: Figs. 1.6, 2.4, 2.8, 4.8, 5.5, and 5.8. The following figures are courtesy of the King's College Library, Cambridge: Figs. 5.6 (AMT/K/7/9), 6.1 (AMT/K/7/14). The following figures are courtesy of the Archives of the Mathematisches Forschungsinstitut Oberwolfach: Figs. 2.2, 2.10, 3.4, 3.7, 4.5, 5.2, and 5.3. Figure 3.6 is courtesy of *Los Alamos National Laboratory Archives,* US. (Unless otherwise indicated, this information has been authored by an employee or employees of the Los Alamos National Security, LLC (LANS), operator of the Los Alamos National Laboratory under Contract No. DE-AC52-06NA25396 with the U.S. Department of Energy. The U.S. Government has rights to use, reproduce, and distribute this information. The public may copy and use this information without charge, provided that this Notice and any statement of authorship are reproduced on all copies. Neither the Government nor LANS makes any warranty, express or implied, or assumes any liability or responsibility for the use of this information.)

I also thank the staff at Springer for all their help with the preparation of this book. In particular, I thank *Ronan Nugent*, my editor at Springer in Heidelberg, for his advice and kind support over the past few years. Finally, I thank the anonymous reviewers for their many valuable suggestions.

Ljubljana, January 2015 *Borut Robič*

Springer

http://www.springer.com/978-3-662-44807-6

The Foundations of Computability Theory

Robič, B.

2015, XX, 331 p. 109 illus., Hardcover
ISBN: 978-3-662-44807-6