Theory and Applications of Computability

In cooperation with the association Computability in Europe

Robert I. Soare

Turing Computability

Theory and Applications

Springer

Robert I. Soare
Department of Mathematics
The University of Chicago
Chicago, Illinois, USA

03Dxx (Computability and recursion theory).

ISSN 2190-619X ISSN 2190-6203 (electronic)
Theory and Applications of Computability
ISBN 978-3-642-31932-7 ISBN 978-3-642-31933-4 (eBook)
DOI 10.1007/978-3-642-31933-4

Library of Congress Control Number: 2016944469

Cover illustration: Damir Dzhafarov designed the image of the Turing machine used in the book cover.

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer-Verlag GmbH Berlin Heidelberg

I dedicate this book to my wife, Pegeen.

Contents

Preface

The title of this book, *The Art of Turing Computability: Theory and Applications,* emphasizes three very important concepts: (1) *computability* (effective calculability); (2) *Turing* or classical computability in the sense of Turing and Post; and (3) the *art* of computability: as a skill to be practiced, but also emphasizing an esthetic sense of beauty and taste in mathematics.

The Art of Classical Computability

Mathematics is an art as well as a science. We use the word "art" in two senses. First "art" means a *skill* or craft which can be acquired and improved by practice. For example, Donald Knuth wrote *The Art of Computer Programming,* a comprehensive monograph in several volumes on programming algorithms and their analysis. Similarly, the present book is intended to be a comprehensive treatment of the *craft* of computability in the sense of knowledge, skill in solving problems, and presenting the solution in the most comprehensible, elegant form. The sections have been rewritten over and over in response to comments by hundreds of readers about what was clear and what was not, so as to achieve the most elegant and easily understood presentation.

However, in a larger sense this book is intended to develop the art of computability as an *artistic endeavor*, and with appreciation of its mathematical beauty. It is not enough to state a valid theorem with a correct proof. We must see a sense of beauty in how it relates to what came before, what will come after, the definitions, why it is the right theorem, with the right proof, in the right place.

One of the most famous art treasures is Michelangelo's statue of *David* displayed in the Accademia Gallery in Florence. The long aisle to approach the statue is flanked with the statues of Michelangelo's unfinished slaves struggling as if to emerge from the block of marble. There are practically no details, and yet they possess a weight and power beyond their physical proportions. Michelangelo thought of himself, not as carving a statue, but as seeing the figure within the marble and then chipping away the marble to release it. The unfinished slaves are perhaps a more revealing example of this talent than the finished statue of David.

Similarly, it was Alan Turing in 1936 and 1939 who saw the figure of computability in the marble more clearly than anyone else. Finding a formal definition for effectively calculable functions was the first step, but *demonstrating* that it captured computability was as much an artistic achievement as a purely mathematical one. Gödel himself had expressed doubt that it would be possible to do so. The other researchers thought in terms of mathematical formalisms like recursive functions, λ-definable functions, and arithmetization of syntax. It was Turing who saw the computer itself in the marble, a simple intuitive device equipped with only a finite program and using only a finite sequence of strokes at each stage in a finite computation, the vision closest to our modern computer. Even more remarkable, Turing saw how to explicitly *demonstrate* that this mechanical device captured all effectively calculable processes. Gödel immediately recognized this achievement in Turing and in no one else.

The first aim of this book is to present the *craft* of computability, but the second and more important goal is to teach the reader to see the figure inside the block of marble. It is to allow the reader to understand the nature of a computable process, of a set which can be computably enumerated, of the process by which one set B is computed relative to another set A, of a method by which we measure the information content of a set, an algebraic structure, or a model, and how we approximate these concepts at a finite stage in a computable process.

The Great Papers of Computability

During the 1930's, educators suggested that college students should read the great books of Western culture in the original. At the University of Chicago the principal proponents were President Robert Maynard Hutchins and his colleague Professor Mortimer Adler. The curriculum relied on *primary* sources as much as possible and a discussion under the supervision of a professor. For decades the Great Books Program became a hallmark of a University of Chicago education.

In the first two decades of Computability Theory from 1930 to 1950 the primary sources were *papers* not books. Most were reprinted in the book by Martin Davis [1965] *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems, and Computable Functions.* Of course, all of these papers are important, shaped the subject, and should be read by the serious scholar. However, many of these papers are written in a complicated mathematical style which is difficult for a beginner to comprehend. Nevertheless, at least two of these papers are of fundamental importance and are easily accessible to a beginning student. My criteria for selecting these papers are the following.

1. The paper must have introduced and developed a topic of fundamental importance to computability.
2. The topic and its development must be as important today as then.
3. The paper must be written in a clear, informal style, so appealing that any beginning student will enjoy reading it.

There are two papers in computability which meet these criteria.

Turing's 1936 Paper, Especially §9

Turing's 1936 paper is probably the single most important paper in computability. It introduces the Turing machine, the universal machine, and demonstrates the existence of undecidable problems. It is most often used in mathematics and computer science to define computable functions. It is perhaps Turing's best known and most influential paper.

I am especially recommending Turing *The Extent of the Computable Numbers,* §9, pp. 249–254 in Turing's 1936 paper. Here Turing gives a demonstration that the numbers computable by a Turing machine "include all numbers which would naturally be regarded as computable." This is a brilliant demonstration and is necessary for the argument. Without it we do not know that we have diagonalized against *all* potential decidable procedures and therefore we have no undecidable problems. Books on computability rarely give this demonstration even though it is critical, perhaps because of its nonmathematical nature. Every student of computability should read this very short section.

Post's 1944 Paper, Especially §11

Turing's 1939 paper very briefly introduced the notion of an "oracle machine," a Turing machine which could consult an oracle tape (database), but he did not develop the idea. In his paper *Recursively Enumerable Sets of Positive Integers and Their Decision Problems,* Emil Post in 1944 developed two crucial ideas, the structure and information content of computably enumerable (c.e.) sets, and the idea of a set B being *reducible* to another set A.

Turing never thought of his oracle machine as a device for reducing one set to another. It was simply a local machine interacting with an external database as today a laptop might query the Internet. Post was the first to turn the oracle machines into a reducibility of a set B to a set A, written $B \leq_{\mathrm{T}} A$, which Post generously called *Turing reducibility.* Post's entire paper is wonderfully written and easily accessible to a beginner. He begins with simpler reducibilities such as many-one reducibility and truth-table reducibility and works up to Turing reducibility which was not understood at the time.

The last section §11 *General (Turing) Reducibility,* is especially recommended. Here Post explored informally the idea of a c.e. set B being Turing reducible to another c.e. set A. For the next decade 1944–1954 Post continued to develop the notions of Turing reducibility and information content. In 1948 Post introduced the idea of *degrees of unsolvability*, now called *Turing degrees*, which are the key to measuring the information content of a set or algebraic structure. Post gave his notes to Kleene before his death in 1954. Kleene revised them and published the Kleene-Post 1954 paper, introducing a finite forcing argument as in Chapter 6 to define Turing incomparable sets. These two notions, *computability* by Turing's automatic machine (a-machine) in 1936, and *reducibility* of one set B to another set A in Turing's 1939 paper and Post's 1944 paper, are the two *most important* ideas in computability theory. Therefore, these papers should be read by anyone taking a course from this book.

Other excellent computability papers are reprinted in [Davis 1965], especially the Gödel Incompleteness Theorem in [Gödel 1931] with the improvement by Rosser. Some of these papers may be difficult for a beginner to read, but they will be more accessible after a first course in computability. Gödel's collected works can be found in the three volumes, [Gödel 1986], [Gödel 1990], and [Gödel 1995].

Introduction

Turing Machines

A Turing machine (a-machine) is a kind of idealized typewriter with an infinite tape and a reading head moving back and forth one cell at a time (§1.4) according to a finite state program. In 1936 Turing demonstrated convincingly that this mathematical model captured the informal notion of effectively calculable. Turing's model and analysis have been accepted ever since as the most convincing model. It is the one on which we base the results in this book.

Oracle Machines and Turing Reducibility

Immediately after this paper, Turing went to Princeton where he wrote a PhD dissertation with Alonzo Church. The dissertation was mainly about ordinal logics, a topic suggested by Church, but one page described an *oracle machine (o-machine)* which is of the greatest importance in computability theory. Turing's oracle machine consisted of a Turing machine connected to an "oracle" which it could query during the computation. This is analogous to the modern model of a local server, such as a laptop computer, connected to a large database, such as the Internet which contains too much information to be stored on the local server.

Turing's oracle machine concept lay dormant for five years until Emil Post's extraordinary 1944 paper revived it, greatly expanded it, and cast the subject in an informal, intuitive light. Post defined a set B to be *Turing reducible* to a set A, written $B \leq_{\mathrm{T}} A$, if there is an oracle machine which computes B when the characteristic function of A is written on the oracle tape. The oracle machines include the ordinary machines because if a set B is computed by an ordinary Turing machine, then it is computed by an oracle machine with $A = \emptyset$ on the oracle tape. But the oracle machines do much more. Turing reducibility is a crucial concept because in computablility theory and applications we rarely prove results about computable functions on computable sets. We compare *noncomputable* (undecidable) sets B and A with respect to their relative information content. We say that sets A and B are *Turing equivalent*, written $A \equiv_{\mathrm{T}} B$ if $A \leq_{\mathrm{T}} B$ and $B \leq_{\mathrm{T}} A$, in which case we view A and B as coding the same information. Turing reducibility gives us a precise measure of the information they encode relative to other sets and the *Turing degrees* (§3.4) are equivalence classes containing sets with the same information content.

Computable Enumerable Sets

In 1936 Church and Kleene introduced the concept of a *computably enumerable* (c.e) set, also called a recursively enumerable (r.e.) set, as one which can be effectively listed, such as the theorems in a formal system like Peano arithmetic. In 1944 Post realized the importance of these sets in many areas of mathematics, and Post devoted much attention to studying their information content. His work on the structure of these sets and their information content under stronger reducibilities has had a great influence on the topics in this book.

Of the effective listing of c.e. sets, $\{W_e\}_{e \in \omega}$ Post reminded us that the Gödel diagonal set $K = \{e : e \in W_e\}$ is c.e. and noncomputable. The famous *Post Problem* was to determine whether there is only one such set up to Turing degree.

Bounded Turing Reductions

At first, Post did not make much progress on the general case of Turing reducibility. To progress toward it, he considered various stronger reducibilities called bounded reducibilities. A Turing reduction $\Phi_e^A = B$ witnessing $B \leq_{\mathrm{T}} A$ is a *bounded Turing reduction*, written $B \leq_{\mathrm{bT}} A$, if there is a computable function $h(x)$ bounding the use function, namely $\varphi_e^A(x) \leq h(x)$, where the use function $\varphi_e^A(x)$ is the maximum element *used* (scanned) during the computation.

For example, every c.e. set B is many-one reducible to K, $B \leq_{\mathrm{m}} K$ by a computable function f, i.e., $x \in B$ iff $f(x) \in K$. Post introduced several structural properties of a c.e. set B in an attempt to prove incompleteness. For example, he proved that a $K \not\leq_{\mathrm{m}} B$ for a simple set B. The varieties of simple and nonsimple sets he introduced and his various bounded reducibilities had a profound effect on the subject for decades and led indirectly to most of the results in this book.

Finally Understanding Turing Reducibility

Post realized that the bounded reducibilities would not solve his problem. It required a deeper understanding of Turing reducibility. His understanding increased over the next decade from 1944 until his death in 1954. Post introduced the notion of *degree of unsolvability* to collect into one equivalence class sets coding the *same information content*. He wrote notes on his work. As he became terminally ill in 1954, Post gave them to Kleene who expanded them and published it as [Kleene and Post 1954]. This paper was a fundamental advance toward solving Post's problem and toward understanding Turing reducibility. The key idea was the continuity of Turing functionals that if $\Phi_e^A(x) = y$ then $\Phi_e^\sigma(x) = y$ for some finite initial segment $\sigma \prec A$, and that if $B \succ \sigma$ then $\Phi_e^B(x) = y$ also.

Using this, Kleene and Post constructed sets A and B computable in K such that $A \not\leq_{\mathrm{T}} B$ and $B \not\leq_{\mathrm{T}} A$. Hence, $\emptyset <_{\mathrm{T}} A <_{\mathrm{T}} K$. This did not explicitly solve Post's Problem because the sets were not c.e., but Kleene and Post divided the conditions into requirements of the form $\Phi_e^A \neq B$, which could be arranged in a priority list of order type ω and processed one at a time using the Use Principle. This became the model for most arguments in the subject. It became the key step in the later solution of Post's Problem by Friedberg in 1957 and Muchnik in 1956 since they combined this strategy with a computable approximation stage by stage. From these ideas emerged the understanding that a Turing functional Φ_e is continuous as a map on Cantor space 2^ω and is not only continuous but *effectively continuous* because the inverse image of a basic open set is the *computable* union of basic open sets.

Priority Arguments

The Kleene-Post construction had produced finite initial segments $\sigma \prec A$ and $\tau \prec B$ such that for some x, $\Phi_e^\sigma(x) \neq \tau(x)$. Hence $\Phi_e^A(x) \neq B(x)$. To make the sets A and B *computably enumerable*, Muchnik and Friedberg had to abandon the K-oracle and computably enumerate the sets, letting A_s be the finite set of elements enumerated in A by the end of stage s and likewise for B_s. They attempted to preserve strings $\sigma \prec A_s$ and $\tau \prec B_s$ when it seemed to give $\Phi_e^\sigma(x) \neq \tau(x)$. This action might later be *injured* because action by a higher priority requirement forces $\sigma \not\prec A_{s+1}$ causing this condition for e to begin all over again. These results led to much more complicated infinite injury arguments.

Other Parts of This Book

The introduction so far explains the background and motivation for most of Part I up to finite injury priority arguments in Chapter 7. For the summary and motivation of the other parts see the next section about how to read this book.

How to Read This Book

Part I: Foundations of Computability

The core of the subject is Part I, Chapters 1–7, from the definition of Turing machines in Chapter 1 up to finite injury priority arguments in Chapter 7. Traditionally, a beginning undergraduate or graduate course of ten or fifteen weeks would go through the sections here one by one. Part I has been streamlined, with more complicated chapters moved to later parts in order to make this schedule feasible. After finishing Chapter 7 on finite injury, the reader will have a firm grasp of the fundamental results and methods of computability theory. One can also cover Part I more quickly as an initial segment of an advanced computability course by concentrating on the starred sections in Part I and then moving to other advanced topics.

Part II: Trees and Π^0_1 Classes

A *tree* is a set of strings closed under initial segments and a Π^0_1 *class* is the set of paths through a computable binary tree. These classes play an important role in model theory, extensions of Peano arithmetic, algorithmic randomness, and other applications. We study open and closed computable classes of reals, and basis and nonbasis theorems for Π^0_1 classes. We give a proof of the Superlow Basis Theorem, proved but not published, by Jockusch and Soare about 1969. We also give a proof by Dzhafarov and Soare of the Low Antibasis Theorem by Kent and Lewis. We show how Π^0_1

classes and their basis theorems are related to models of Peano arithmetic, with results by Jockusch and Soare, Scott, Shoenfield, and Solovay. Finally, we relate Π^0_1 classes to Martin-Löf randomness, computably dominated (hyperimmune-free) degrees, and to computably traceable sets.

Part III: Minimal Degrees

A Turing degree $\mathbf{a}$ is *minimal* if $\mathbf{a} > \mathbf{0}$ but there is no degree $\mathbf{b}$ such that $\mathbf{0} < \mathbf{b} < \mathbf{a}$. In Chapter 12 we present Spector's proof of a minimal degree $\mathbf{a} < \mathbf{0}''$. The proof uses a forcing argument like those in Chapter 6 but with more complicated forcing conditions of perfect trees instead of finite strings. In Chapter 13 we present the Sacks construction of a minimal degree $\mathbf{a} < \mathbf{0}'$. This is an approximation to Spector's method and uses a finite injury priority argument. We also sketch a limit computable (full approximation) construction of a minimal degree below $\mathbf{0}'$ using a computable construction. It can also be done below any nonzero computably enumerable degree, thereby producing a low minimal degree. Chapters 6 and 7 are the only prerequisites for this material.

Part IV: Games in Computability Theory

Games are very important in understanding the nature of computability, how to prove theorems, and how to solve problems. In Chapter 14 we present the classical Banach-Mazur games which are closely related to the finite extension constructions of Chapter 6, Sections 1–3, and may be read simultaneously with them and with Chapter 8 on open and closed classes in Cantor space. Players I and II alternately construct strings σ_{2n} and σ_{2n+1}, jointly constructing a point $f = \cup_n \sigma_n$ in Cantor space 2^ω. There is a predetermined class $\mathcal{A} \subseteq 2^\omega$. Player I wins the game according to whether $f \in \mathcal{A}$ or not. Winning strategies are described in terms of properties of $\mathcal{A}$. We also discuss the Cantor-Bendixson rank of points in a closed subclass $\mathcal{A} \subseteq 2^\omega$.

In Chapter 15 we make a very brief excursion into Gale-Stewart games. We consider the complexity of the winning strategy for a very simple computable game.

We finish in Chapter 16 by returning to the topic of more Lachlan games, first introduced in §2.5. These games are the *principal tool* in proving theorems and solving problems in computability theory.

Symbols Marking Importance and Difficulty

In Part I we use the following notation for sections, theorems, and exercises.

$\star\star$	Most important.
$\star$	Very important.
No Marking	Average importance.
$\oslash$	Skim or defer on a first reading until needed in a later chapter.
$\diamond$	Difficult exercise, do not assign lightly.
$\diamond\diamond$	Very difficult exercise.

Notation

Notation will be defined when introduced. We now summarize the most common notation and definitions.

Notation for Sets

The universe is the set of nonnegative integers $\omega = \{0, 1, 2, 3, \dots\}$ which sometimes appears in the literature as $\mathbb{N}$. Most of the objects we study can be associated with some $n \in \omega$ called a "code number" or "Gödel number." We can think of operations on these objects as being presented by a corresponding function on these numbers and our functions will have domain and range contained in ω.

Uppercase Latin letters A, B, C, D and X, Y, Z normally represent subsets of $\omega = \{0, 1, 2, 3, \dots\}$ with the usual set operations $A \cup B$, $A \cap B$; $|A|$, or $\mathrm{card}(A)$ denotes the cardinality of A; $\max(A)$ denotes the maximum element $x \in A$ if A is finite; $A \subseteq B$ denotes that A is a subset of B, and $A \subset B$ that it is a *proper* subset; $A - B$ denotes the set of elements in A but not in B; $\overline{A} = \omega - A$, the complement of A; $A \sqcup B$ denotes the *disjoint union*, i.e., $A \cup B$ provided that $A \cap B = \emptyset$; the *symmetric difference* is $A \,\Delta\, B = (A - B) \cup (B - A)$; a, b, c, $\dots x$, y, z, $\dots$ represent integers in ω; $A \times B$ is the Cartesian product of A and B, the set of ordered pairs (x, y) such that $x \in A$ and $y \in B$; $\langle x, y\rangle$ is the integer that is the image of the pair (x, y) under the standard pairing function from $\omega \times \omega$ onto ω; $A \subseteq^* B$ denotes that $|A - B| < \infty$; $A =^* B$ denotes that $A \,\Delta\, B$ is finite; $A \subset_\infty B$

denotes that $|B - A| = \infty$. Given a simultaneous enumeration (see p. 46) of A and B let $A \setminus B$ denote the set of elements enumerated in A before B and $A \searrow B = (A \setminus B) \cap B$, the set of elements appearing in A and later in B.

Logical Notation

We form predicates with the usual notation of logic where &, $\vee$, $\neg$, $\Longrightarrow$, $\exists$, $\forall$ denote respectively, and, or, not, implies, there exists, for all; $(\mu x)\, R(x)$ denotes the least x such that $R(x)$ if it exists, and is undefined otherwise; $(\exists^\infty x)$ denotes "there exist infinitely many x," and $(\forall^\infty x)$ denotes "for almost all x" as in Definition 3.5.1. These quantifiers are dual to each other. The latter is written $(\exists x_0)(\forall x \geq x_0)$. We use $x, y, z < w$ to abbreviate $x < w,\ \ y < w$, and $z < w$. In a partially ordered set we let $x \mid y$ denote that x and y are incomparable, i.e., $x \not\leq y$ and $y \not\leq x$. We often use the dot convention to abbreviate brackets before and after the principal connective of a logical expression. For example, if α and β are well-formed formulas, then $\alpha \ . \ \Longrightarrow \ . \ \beta$ abbreviates $[\ \alpha\ ] \ \Longrightarrow \ [\ \beta\ ]$. The algorithm is to insert a right bracket just before $\Longrightarrow$ and then a matching left bracket just before the first symbol in α. Do the corresponding algorithm for β. The dots increase readability of a long expression. TFAE abbreviates "The following are equivalent."

We use the usual Church *lambda notation* for defining partial functions. Suppose $[\ldots x \ldots]$ is an expression such that for any integer x the expression has at most one corresponding value y. Then $\lambda x\, [\ldots x \ldots]$ denotes the associated partial function $\theta(x) = y$, for example $\lambda x\, [\, x^2\,]$. The expression $\lambda x\, [\uparrow]$ denotes the partial function which is undefined for all arguments. We also use the lambda notation for partial functions of k variables, writing $\lambda x_1 x_2 \ldots x_k$ in place of λx. An expression such as $\lambda x\, y\, [\, x + y\,]$, denotes addition as a function of x and y. However, $\lambda x\, [\, x + y\,]$ indicates that the expression is viewed as a function of x with y as a parameter, such as $\lambda x\, [x + 2]$. One advantage is that with an expression of several arguments, such as in the *s-n-m* Theorem 1.5.5 (Parameter Theorem) we can make clear which arguments are variables and which are parameters, for example as explained in Remark 1.5.6. Define $f(x) = 1 \dot{-} x$ to be 1 if $x = 0$ and 0 if $x \geq 1$. We call this the *monus* function. It produces a 0-1 valued function $f(x) \neq x$.

Lattices and Boolean Algebras

A *lattice* $\mathcal{L} = (L; \leq, \vee, \wedge)$ is a partially ordered set (poset) in which any two elements a and b have a least upper bound (lub) $a \vee b$ and greatest lower bound (glb) $a \wedge b$. An *upper semi-lattice* has lub only. For example, the Turing degrees under Turing reducibility form an upper semi-lattice. If $\mathcal{L}$ contains a least element and greatest element these are called the *zero* element 0 and *unit* element 1, respectively. In such a lattice a is the *complement* of b if $a \vee b = 1$ and $a \wedge b = 0$, and $\mathcal{L}$ forms a *Boolean algebra* if every element has a complement. A nonempty subset $\mathcal{I} \subseteq \mathcal{L}$ forms an *ideal* $\mathcal{I} = (I; \leq, \wedge, \vee)$ of $\mathcal{L}$ if $\mathcal{I}$ satisfies the conditions:

(1) $[a \in L \ \& \ a \leq b \in I] \implies a \in I$, and

(2) $[a \in I \ \& \ b \in I] \implies a \vee b \in I.$

A *filter* $F \subset L$ satisfies the dual conditions. For example, the subsets of ω form a Boolean algebra with the finite sets as an ideal and the cofinite sets as a filter.

Notation for Strings and Functionals

We let $2^{<\omega}$ denote the set of all finite sequences of 0's and 1's called strings and denoted by σ, ρ, and τ. Let 2^{ω} denote the set of all functions f from ω to $2 = \{0, 1\}$, and ω^{ω} the set of all functions f from ω to ω. The integers $n \in \omega$ are *type 0* objects, (partial) functions $f \in 2^{\omega}$ or subsets $A \subseteq \omega$ (which are identified with their characteristic function $\chi_A \in 2^{\omega}$) are *type 1* objects, a (partial) *functional* Ψ is a map from type 1 objects to type 1 objects, i.e., a map from 2^{ω} to 2^{ω} and is called a *type 2* object. Identifying a set A with its *characteristic function* χ_A we often write $A(x)$ for $\chi_A(x)$. Uppercase Latin letters A, B, C, ..., represent subsets of ω. Script letters $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ represent subsets of 2^{ω} and are called *classes* to distinguish them from sets.

Gödel Numbering of Finite Objects

In his Incompleteness Theorem [1931] Gödel introduced the method of assigning a *code number* or *Gödel number* to every formal (syntactical) object such as a formula, proof, and so on. We now present two ways to effectively code a sequence of n-tuples of integers $\{a_1, a_2, \dots a_n\}$, define

$$(1) \qquad a \ = \ p_1^{a_1+1} \, p_2^{a_2+1} \ \dots \ p_k^{a_k+1}$$

where p_i is the ith prime number. Given a we can effectively recover the prime power $(a)_i = a_i + 1$. This coding is injective but not surjective on ω.

The second method uses the following standard pairing function and has the added advantage that the n-tuple coding below is an injective and surjective map from ω onto ω^n.

Standard Pairing Function

(i) Let $\langle x, y\rangle$ denote the integer that is the image of the ordered pair (x, y) under the standard pairing function $\frac{1}{2}(x^2 + 2xy + y^2 + 3x + y)$ which is a 1:1 computable function from $\omega \times \omega$ onto ω. Let π_1 and π_2 denote the *inverse pairing functions* $\pi_1(\langle x, y\rangle) = x$, and $\pi_2(\langle x, y\rangle) = y$.

(ii) Let $\langle x_1, x_2, x_3\rangle$ denote $\langle\langle x_1, x_2\rangle, x_3\rangle$. Let the *$n$-ary pairing function* be

$$(2) \qquad \langle x_1, x_2, \ldots, x_n\rangle \quad = \quad \langle \cdots \langle\langle x_1, x_2\rangle, x_3\rangle, \ldots, x_n\rangle.$$

(All these functions are clearly computable and even primitive recursive.)

If the sequences are all of fixed length n we may use method 2, the n-ary pairing function of (2),

$$(3) \qquad f(a_1, a_2, \ldots a_n) \;=\; \langle a_1, a_2, \ldots a_n\rangle$$

Otherwise, we use the first method above of coding using prime powers. There are many other coding algorithms. The important point for coding is that the method be effective and invertible, but it is often useful to have it surjective as well.

Note that both methods are effectively invertible. Let θ be any 1:1 computable partial function. Then θ is effectively invertible on its range. Just enumerate the pairs (u, v) with $\theta(u) = v$ until, if ever, a pair (x, y) is found and then define $\psi(y) = x$. See also the Definition 2.1.7 of graph(θ) and the Uniformization Theorem 2.1.8.

Effective Numbering of Finite Sets and Strings

Given a finite set $F = \{x_1, x_2, \ldots x_k\}$ where $x_1 < x_2 \ldots x_k$ we give F the (strong) index $y = 2^{x_1} + 2^{x_2} \ldots + 2^{x_k}$ and write that $D_y = F$. Let$D_0 = \emptyset$. Likewise, give every string $\sigma \in 2^\omega$ an effective index from using either the strong index coding or Gödel numbering so that from the index we can recover the length $k = |\sigma|$ and every component $\sigma(i)$, for $i < k$. Such a numbering of strings σ_z is given in Definition 2.3.6.

Partial Computable (P.C.) Functions

Let $\{P_e\}_{e\in\omega}$ be an effective numbering of all Turing machine programs (as in Definition 1.5.1). We write $\varphi_e(x) = y$ if program P_e with input x halts and yields output y, in which case we say that $\varphi_e(x)$ *converges* (written $\varphi_e(x)\downarrow$), and otherwise $\varphi_e(x)$ *diverges* (written $\varphi_e(x)\uparrow$); $\{\varphi_e\}_{e\in\omega}$ is an effective listing of all *partial computable (p.c.)* functions; the domain and range of φ_e are denoted by dom(φ_e) and rng(φ_e). A set A is *computably enumerable (c.e.)* if it can be effectively listed, i.e., if A = dom(φ_e) for some e.

If $\mathrm{dom}(\varphi_e) = \omega$ then φ_e is a *total computable function* (abbreviated *computable function*); we let f, g, h, ... denote total functions; $f \circ g$ or fg denotes the composition of functions, applying first g to an argument x and then applying f to $g(x)$. Let $f \upharpoonright x$ denote the restriction to elements $y < x$ and $f \upharpoonright\upharpoonright x$ the restriction to elements $y \leq x$.

Turing Functionals Φ_e^A

Let $\{\widetilde{P}_e\}_{e\in\omega}$ be an effective numbering of all Turing machine oracle programs, finite sets of sextuples defined in §3.2.1. Write $\Phi_e^A(x) = y$ if oracle program $\widetilde{P}_e$ with A on its oracle tape and input x halts and yields output y. Let the use function $\varphi_e^A(x)$ be the greatest element z for which the computation scanned the square $A(z)$ on the oracle tape. We regard Φ_e as a (partial) functional (type 2 object) from 2^ω to 2^ω mapping A to B if $\Phi_e^A = B$.

The use function $\varphi_e^A(x)$ has an exponent A to distinguish from the p.c. function $\varphi(x)$. They usually come in matched pairs, $\Psi^A(x)$ and $\psi^A(x)$, $\Theta^A(x)$ and $\theta^A(x)$, where the lowercase function denotes the use function corresponding to the uppercase functional. See Definition 3.2.2 (vi) for a function f as oracle in place of the set A.

Lachlan Notation

When $E(A_s, x_s, y_s, \ldots)$ is an expression with a number of arguments subscripted by s denoting their value at stage s, Lachlan introduced the notation $E(A, x, y, \ldots)[\,s\,]$ to denote the evaluation of E where all arguments are taken with their values at the end of stage s.

(4) $\Phi_e^A(x)\,[\,s\,]$ denotes $\Phi_{e,s}^{A_s}(x_s)$ and $\varphi_e^A(x)\,[\,s\,]$ denotes $\varphi_{e,s}^{A_s}(x_s)$.

This Lachlan notation has become very popular and is now used in most papers and books.

Acknowledgements

Among others I would like to thank colleagues and students for careful reading of preliminary versions of this book and for their suggestions and corrections.

This includes my University of Chicago colleagues, Denis Hirschfeldt, Joseph Mileti, Antonio Montalban, and Maryanthe Malliaris; and former students, Eric Astor, William Chan, Barbara Csima, Chris Conidis, David Diamondstone, Damir Dzhafarov, Rachel Epstein, Kenneth Harris, Karen Lange, Russell Miller, Jonathan Stephenson, and Matthew Wright.

It includes colleagues at other universities, Ted Slaman, Richard Shore, Carl Jockusch, Douglas Cenzer, Leo Harrington, Manuel Lerman; and colleagues at the University of Wisconsin, Steffen Lempp, Bart Kastermans, Arnie Miller, Joe Miller, and Wisconsin students, Asher Kach, Dan Turetsky, and Nathan Collins.

I am grateful to the students of Barbara Csima at the University of Waterloo, Vladimir Soukharev, Jui-Yi Kao, Atul Sivaswamy, David Belanger, and Carolyn Knoll; to Piet Rodenburg and Tom Sterkenbert in Amsterdam and their students, including Frank Nebel; to Notre Dame colleagues Julia Knight and Peter Cholak and their students, Joshua Cole, Yang Lu, Stephen Flood, Quinn Culver and John Pardo; to Valentina Harizanov at George Washington University and her students, Jennifer Chubb and Sarah Pingrey; to Aaron Sterling at Iowa State University; to Russell Miller's student at Queens College CUNY, Rebecca Steiner; to Iraj Kalantari at Northern Illinois University and his student, Abolfazi Karimi; to Rachel Epstein and her students at Harvard who covered Part I of the book in detail; to Damir Dzhafarov who drew the diagrams in tikz; to Linda Westrick at the University of Connecticut. Carl Jockusch and Damir Dzhafarov read some of the advanced chapters in detail and made a number of mathematical corrections and suggestions. Damir also designed the excellent cover diagram of a Turing machine.

I am grateful to Ronan Nugent, Senior Editor at Springer-Verlag, who read the manuscript in detail, made a number of corrections, and handled the editing and production of this book.