Preface

Computable analysis is a branch of computability theory studying those functions on the real numbers and related sets which can be computed by machines such as digital computers. The increasing demand for reliable software in scientific computation and engineering requires a sound and broad foundation not only of the analytical/numerical but also of the computational aspects of real number computation. Although many researchers have been active in computable analysis, it has never belonged to the main stream of research in computability. Our knowledge of this field is remarkably insufficient and only very few mathematicians or computer scientists know a definition of computable real functions. At present, computable analysis appears as a juxtaposition of several partly independent approaches which are more or less developed. For the interested newcomer this situation is bewildering, since there are not even generally accepted basic definitions; therefore, learning the state of the art from the fragments is a laborious undertaking.

This book is a new attempt to present a coherent basis for computable analysis. It is intended as a textbook suitable for graduate students in computer science or mathematics. Merely elementary knowledge in computability and analysis is assumed as prerequisite. Although many parts of the book offer themselves for extension or generalization, I have tried to concentrate on the most important elementary topics and to remain at a homogeneous moderate "level of abstraction" in order to keep the text short and make it accessible to a broader readership.

The central subject of the book is "Type-2 Theory of Effectivity" (TTE), one of the approaches to effective analysis being discussed today. It is based on definitions of computable real numbers and functions by A. Turing [Tur36] A. Grzegorczyk [Grz55] and D. Lacombe [Lac55]. Basic concepts of TTE are explained informally in Section 1.3. Chapters 2–7 systematically develop foundations of TTE. A framework of "concrete" computability on finite and infinite sequences of symbols is introduced in Chapter 2. Computability on finite and infinite sequences of symbols can be transferred to other sets by using them as names. Computability induced by naming systems is discussed in Chapter 3, where, in particular, the important class of "admissible representations" is introduced. Chapter 4 is devoted to computable real numbers and functions. Computability on spaces of subsets of $\mathbb{R}^n$ and on spaces of real functions are introduced and discussed in Chapters 5 and 6, respectively.

As a refinement of computability, the computational complexity of real functions is introduced in Chapter 7. Computable metric spaces and degrees of discontinuity are extensions of the basic theory which are discussed in Chapter 8. Finally, in Chapter 9 some other approaches to computable analysis are introduced and compared with TTE.

Most sections end with a number of exercises which particulary provide the instructor with material for homework and tests. Unmarked exercises are of medium difficulty. Exercises marked by $\Diamond$ are easy and may be solved in a straightforward manner with a proper understanding of the text. Exercises marked by $\blacklozenge$ are difficult or require a trick to solve. Such a rating of difficulty is, of course, subjective. The reader should make every effort to solve the exercises, at least the easier ones. Similarly, in the text itself, the reader should attempt to prove theorems, whenever possible, without first reading the proof in the text. Many exercises are extensions or generalizations of the material presented in the main text. Throughout the book, the square $\square$ denotes the end of a proof or example.

Since discussion on models of computation and the most adequate concepts and tools for computable analysis is still ongoing, I have put an emphasis on explaining and discussing the central definitions in detail and pointing out their distinctive features.

The origins of this book lie in a number of research papers, and more concretely in Part 3 of a monograph [Wei87], a correspondence course [Wei94] and a technical report [Wei95]. While still writing it, some gaps appeared in the subject which had to be filled, and therefore the book contains material not yet published elsewhere. However, numerous important questions in computable analysis, and even many elementary ones, are still unsettled and await systematic exploration. Although many references are included, the list is far from being complete. I apologize to all those whose work is insufficiently or not mentioned.

I should like to thank the students and collaborators who have contributed to the development of TTE, in particular U. Schreiber, G. Schäfer, C. Kreitz, T. Deil, N. Müller, T. v. Stein, U. Mylatz, M. Schröder, V. Brattka, P. Hertling, X. Zheng and N. Zhong. I have benefited from the discussions with many friends and colleagues, and I am especially grateful to V. Brattka, P. Hertling, M. Schröder, X. Zheng, N. Zhong, J. Zucker and several other people, last but not least the Springer-Verlag copy editors, who have read preliminary versions of the text and made many helpful corrections and suggestions. The research would not have been so efficient without the support of two projects of the DFG (Deutsche Forschungsgemeinschaft) under grants We843/3 and We843/8.

Finally, a big thank to my wife, Susanne, for her patience and encouragement during the numerous phases of writing of this book.

August, 2000 Klaus Weihrauch

Contents

1. Introduction

1.1 The Aim of Computable Analysis

All over the world numerous computers are used for real number computation. They evaluate real functions, find zeroes of functions, determine eigenvalues and integrals and solve differential equations. They perform, or at least are expected to perform, computations on sets like $\mathbb{R}$ (the set of real numbers), $\mathbb{R}^n$, $\mathcal{O}(\mathbb{R})$ (the open subsets of real numbers), $\mathcal{K}(\mathbb{R}^n)$ (the compact subsets of $\mathbb{R}^n$) or $\mathrm{C}[0;1]$ (the continuous functions from the real unit interval to the real numbers). The increasing demand for reliable as well as fast software in scientific computation and engineering requires a sound and broad foundation. We agree with L. Blum et al. [BCSS96] (also in [BCSS98], however, see Sect. 9.7):

> Our perspective is to formulate the laws of computation. Thus, we write not from the point of view of the engineer who looks for a good algorithm which solves his problem at hand, or wishes to design a faster computer. The perspective is more like that of a physicist, trying to understand the laws of scientific computation. Idealizations are appropriate, but such idealizations should carry basic truths.
>
> Scientific computation is the domain of computation which is based mainly on the equations of physics. For example, from the equations of fluid mechanics, scientific computation helps to understand better design for airplanes, or assists in weather prediction. The theory underlying this side of computation is called numerical analysis.
>
> There is a substantial conflict between theoretical computer science and numerical analysis. These two subjects with common goals have grown apart. For example, computer scientists are uneasy with calculus, while numerical analysis thrives on it. On the other hand numerical analysts see no use for the Turing machine.
>
> The conflict has its roots in another age-old conflict, that between the continuous and the discrete. Computer science is oriented by the digital nature of machines and by its discrete foundations given by Turing machines. For numerical analysis systems of equations, and differential equations are central and this discipline depends heavily on the continuous nature of the real numbers. [...] Algorithms are primarily a means to solve practical problems. There is not even a formal definition of algorithm in the subject. [...] Thus, we view numerical analysis as an eclectic subject with weak foundations; this certainly in no way denies the great achievements through the centuries.

For a deep understanding and for future development of computation in analysis, a sound theoretical foundation is indispensable. In this book Computable Analysis is developed as the theory of those functions on the real numbers and other sets from analysis, which can be computed by machines. Computable analysis connects the two classical disciplines analysis/numerical analysis and computability/complexity theory. It merges concepts from both of them, in particular, the central concepts of limit and approximation on the one hand and of machine models and discrete computation on the other hand.

1.2 Why a New Introduction?

While analysis and numerical analysis have a very long tradition (mathematicians like Gauß or Lagrange were experts in numerical calculation), it was not until the 1930s that S. Kleene, A. Church, A. Turing and others proposed various definitions of *effectively calculable* functions on the natural numbers, all of which turned out to be equivalent. Meanwhile for functions on the natural numbers or on finite words there exists a well-established and very rich theory of computability and computational complexity [Rog67, HU79, Wei87, Odi89]. Although a number of authors also studied computability on the real numbers, computable analysis is still underdeveloped. In contrast to ordinary computability theory there are several partially non-equivalent suggestions of how to model effectivity in analysis and, in particular, computability of real functions. Even today no theory of computable analysis has been accepted by the majority of mathematicians or computer scientists.

The first author who introduced computable real numbers was A. Turing in his famous article "On computable numbers, with an application to the Entscheidungsproblem" [Tur36, Tur37]. Since that time computable analysis developed continuously. Among the large number of publications there are also some books on computable analysis or closely related topics, for example, R. L. Goodstein[Goo59], D. Klaua [Kla61], S. Mazur [Maz63], O. Aberth [Abe80], B. Kushner [Kuš84], E. Bishop and D. Bridges [BB85], K. Weihrauch [Wei87], M. Pour-El and J. Richards [PER89], J. Traub, G. Wasilkowski and H. Woźniakowski [TWW88], K. Ko [Ko91] and L. Blum, F. Cucker, M. Shub, S. Smale [BCSS98]. While for mathematical branches like linear algebra or recursion theory there are canonical foundations and well established introductions, all these books have important concepts in common, but differ in their contents and technical framework, and each author presents the topic from his individual point of view. This mirrors the fact that computable analysis still has no generally accepted foundation.

This book is a new attempt to present a coherent foundation of computable analysis. It is rooted in a definition of computable real functions via representations introduced by A. Grzegorczyk [Grz55] and later work on the

theory of representations by J. Hauck [Hau73, Hau78, Hau80, Hau81, Hau82] and others. To distinguish it from other approaches, it will be called "Type-2 Theory of Effectivity", TTE, for short.

1.3 A Sketch of TTE

Type-2 Theory of Effectivity extends ordinary (Type-1) computability and complexity theory. We note already here that TTE admits two levels of effectivity, continuity and computability as a specialization of continuity. It is applicable to a variety of problems from analysis and provides a common framework for combining approximation, computation and computational complexity. TTE still allows a variety of computability concepts on the real numbers and other sets. We will select the seemingly most important ones and point out their distinctive features. Before we start developing TTE in detail, in this section we explain some essential ideas and concepts informally, in particular, without using a mathematically precise model of computation.

1.3.1 A Model of Computation

Ordinary computability theory first introduces computable partial word functions $f :\subseteq \Sigma^* \to \Sigma^*$ explicitly, for example, by means of Turing machines. For defining computability on other sets M (rational numbers, finite graphs, etc.) words are used as "names" or "codes" of elements of M. While a machine still transfers words to words, the user interprets these words as names of elements from the set M. Equivalently, one can start also with the computable number functions $f :\subseteq \mathbb{N} \to \mathbb{N}$ and use numbers as names. Since the sets Σ^* and $\mathbb{N}$ are only countable, they are not sufficient as sets of names for the uncountable set $\mathbb{R}$ of real numbers.

However, real numbers can be represented by infinite sequences, for example by infinite decimal fractions (example: `3•14159`$\ldots$ is a name of π). In TTE, infinite sequences are used as names of real numbers, and machines which transfer infinite sequences to infinite sequences are used to compute real functions. Obviously, this method of defining computable functions is not restricted to the real numbers and can be applied to many other sets. Fig. 1.1 shows a machine transforming infinite sequences to infinite sequences.

Fig. 1.1. A machine transforming infinite sequences

On input $(I_0, I_1, I_2, \ldots)$, from time to time the machine reads a new sequence element I_k from its input file and from time to time it writes a new sequence element J_m to its output file, where I_{k+1} is read after I_k and J_{m+1} is written after J_m.

1.3.2 A Naming System for Real Numbers

Since infinite decimal fractions are commonly used for representing real numbers, it seems to be natural to take them as inputs and outputs for machines. However, we will prove later that the induced computability concept on the set $\mathbb{R}$ of real numbers is not very interesting. Instead of infinite decimal fractions we will use sequences of nested intervals as names. In this section, a *name* of a real number $x \in \mathbb{R}$ is a sequence $(I_0, I_1, I_2, \ldots)$ of closed intervals $[a; b]$ with rational endpoints $a < b$, *rational intervals* for short, such that $I_{n+1} \subseteq I_n$ for all $n \in \mathbb{N}$ and $\{x\} = \bigcap_{n\in\mathbb{N}} I_n$. We assume tacitly that intervals are encoded appropriately, such that, strictly speaking, a name of a real number is an infinite sequence of symbols. Fig. 1.2 shows the first six intervals of a name of a real number x.

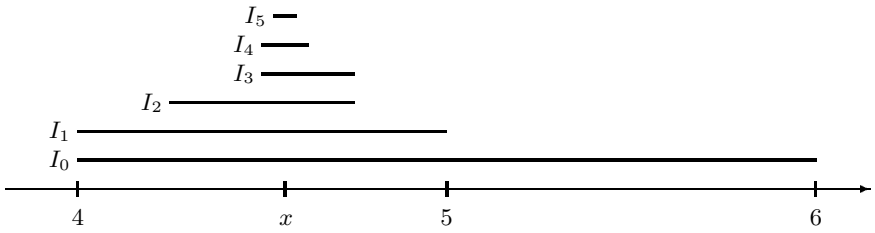

Fig. 1.2. The first elements of a name of x

1.3.3 Computable Real Numbers and Functions

We call a real number *computable*, iff it has a computable name. We illustrate these definitions by examples.

Example 1.3.1 (computable real numbers).

1. Every rational number $r \in \mathbb{Q}$ is computable.
   Define $I_n := [r-2^{-n}; r+2^{-n}]$ for all $n \in \mathbb{N}$. Then the sequence $(I_0, I_1, \ldots)$ is a computable name of r.
2. $\sqrt{2}$ is computable.
   Define $f : \mathbb{N} \to \mathbb{N}$ by $f(n) := \min\{k \in \mathbb{N} \mid k^2 < 2n^2 \leq (k+1)^2\}$, $J_0 := [1; 2]$ and $J_n := [f(n)/n; (f(n)+2)/n]$ for all $n > 0$. Then the sequence $(J_0, J_1, \ldots)$ is computable, $\sqrt{2} \in J_n$ for all n and $\lim_{n\to\infty} \text{length}(J_n) = 0$. But the sequence is not nested in general. Define $I_n := J_0 \cap J_1 \cap \ldots \cap J_n$. Then the sequence $(I_0, I_1, I_2, \ldots)$ is a computable name of $\sqrt{2}$.

3. $\log_3 5$ is computable.
   Define $f(n) := \min\{k \in \mathbb{N} \mid 3^k < 5^n \leq 3^{k+1}\}$ and continue as above. □

Example 1.3.2 (Specker sequences). For $A \subseteq \mathbb{N}$

$$x_A := \sum_{i \in A} 2^{-i} \text{ is computable } \iff A \text{ is recursive}$$

Assume that A is recursive. Define $I_n := [s_n\,;\, s_n + 2 \cdot 2^{-n}]$ where $s_n := \sum_{i \leq n, i \in A} 2^{-i}$. Then $(I_0, I_1, \ldots)$ is a computable name of x_A.

On the other hand, let $(I_0, I_1, \ldots)$ be a computable name of x_A. If A is finite or co-finite, then A is recursive. Assume that A is neither finite nor co-finite. Then $m/2^n \neq x_A$ for all $m, n \in \mathbb{N}$. For $n = 0, 1, \ldots$ (in this order) decide whether $n \in A$ as follows. Define $A_n := \{i \in A \mid i < n\}$ and $t_n := \sum_{i \in A_n} 2^{-i} + 2^{-n}$. Notice that $0 < \sum_{i \in A, i > n} 2^{-i} < 2^{-n}$. Then $n \in A \Longrightarrow x_A > t_n$ and $n \notin A \Longrightarrow x_A < t_n$. Find the smallest $k \in \mathbb{N}$ with $t_n \notin I_k$. If $\max(I_k) < t_n$ then $x_A < t_n$, hence $n \notin A$, if $t_n < \min(I_k)$ then $t_n < x_A$, hence $n \in A$. Therefore, A is recursive.

Let $A \subseteq \mathbb{N}$ be recursively enumerable but not recursive. Then the real number $x_A := \sum_{i \in A} 2^{-i}$ is not computable. From recursion theory we know that $A = \text{range}(f)$ for some computable injective function $f : \mathbb{N} \to \mathbb{N}$. We obtain $x_A = \sum_{i \in \mathbb{N}} 2^{-f(i)}$. Then $(x_0, x_1, \ldots)$ with $x_n = \sum_{i \leq n} 2^{-f(i)}$ is an increasing and bounded computable sequence of rational numbers. Its limit, however, is the non-computable real number x_A. Sometimes such a sequence is called a *Specker sequence* [Spe49]. □

We call a real function $f :\subseteq \mathbb{R} \to \mathbb{R}$ *computable*, iff some machine maps any name of any $x \in \text{dom}(f)$ to a name of $f(x)$. For real functions $f :\subseteq \mathbb{R}^n \to \mathbb{R}$ we consider machines reading n names in parallel. Notice that the machine must behave correctly only for every name of every $x \in \text{dom}(f)$, for other input sequences it may behave arbitrarily.

Example 1.3.3 (computable real functions).

1. Real multiplication $(x, y) \mapsto x \cdot y$ is computable.
   For closed intervals I, J with rational endpoints define the interval $I \cdot J$ by $I \cdot J := \{x \cdot y \mid x \in I, y \in J\}$. There is a machine with two input files which maps the inputs $(I_0, I_1, \ldots)$ and $(J_0, J_1, \ldots)$ to $(I_0 \cdot J_0, I_1 \cdot J_1, \ldots)$. If $(I_0, I_1, \ldots)$ is a name of x and $(J_0, J_1, \ldots)$ is a name of y then $(I_0 \cdot J_0, I_1 \cdot J_1, \ldots)$ is a name of $x \cdot y$. Therefore, multiplication is computable.
2. The square root $\sqrt{\ } :\subseteq \mathbb{R} \to \mathbb{R}$ is computable.
   Since for example $\sqrt{[2;3]}$ is not a rational interval, we have to modify the above method. We map each input interval I to a rational interval which is slightly longer than $\sqrt{I}$. For any rational numbers a, b with $0 \leq a < b$ there are rational numbers $r, s \geq 0$ such that

$$a - (b - a) < r^2 \leq a < b \leq s^2 < b + (b - a)\,.$$

Therefore, there is a computable function f which for each rational interval I with no negative elements determines (for example by exhaustive search) a rational interval $K = f(I)$ such that $I \subseteq K^2$ and $\text{length}(K^2) < 3 \cdot \text{length}(I)$. For any rational interval $[a;b]$ with $0 \leq b$ let $g[a;b] := [\max(0,a);b]$. If $(I_0, I_1, \ldots)$ is a name of $x \geq 0$ then the sequence $(f \circ g(I_0), f \circ g(I_1), \ldots)$ of rational intervals converges to $\sqrt{x}$. This sequence is not necessarily nested. There is a machine M which transforms any name $(I_0, I_1, \ldots)$ of a non-negative real number to the sequence $(J_0, J_1, \ldots)$, where $J_n := f \circ g(I_0) \cap f \circ g(I_1) \cap \ldots \cap f \circ g(I_n)$. Then M computes the square root. □

We will prove later that elementary real functions like $\exp, \sin, \log$ and arcsin are computable, that the computable real functions map computable numbers to computable numbers and that the computable real functions are closed under composition. Our definition is essentially the definition of computable real function introduced by A. Grzegorczyk in [Grz55] and studied in more detail in [Grz57], where he proves, in particular, that computable real functions are continuous.

Theorem 1.3.4. Every computable real function is continuous.

Proof: We sketch a proof for the case $f : \mathbb{R} \to \mathbb{R}$. Let M be a machine computing f and let $x \in \text{dom}(f) = \mathbb{R}$. It suffices to show that for every open set $V \subseteq \mathbb{R}$ with $f(x) \in V$, there is some open set $U \subseteq \mathbb{R}$ with $x \in U$ and $f[U] \subseteq V$. Therefore, let V be an open set with $f(x) \in V$. The real number x has a name $([a_0;b_0], [a_1;b_1], \ldots)$ such that $a_i < a_{i+1} < b_{i+1} < b_i$. Let $(J_0, J_1, \ldots)$ be the name of $f(x)$ which M produces on this input. Then $J_n \subseteq V$ for some number n. For producing the initial part $(J_0, J_1, \ldots, J_n)$ of the output, the machine M needs k steps for some k. In k steps the machine M can read at most the first k intervals from the input. We choose $U := (a_k; b_k)$. Obviously, we have $x \in U$. Assume $y \in U$. Then y has a name of the form $([a_0;b_0], [a_1;b_1], \ldots, [a_k;b_k], L_{k+1}, L_{k+2}, \ldots)$. On this input, the machine M writes also $(J_0, J_1, \ldots, J_n)$ within the first k steps which is the initial part of a name of $f(y)$. We obtain $f(y) \in J_n$. This shows $f[U] \subseteq J_n \subseteq V$. Therefore, the function f is continuous at x. □

In the above proof we have used the essential observation that any finite portion $(J_0, J_1, \ldots, J_n)$ of the output of a computation is determined already by a finite portion $(I_0, I_1, \ldots, I_k)$ of its input. However, we did not use that the transformation from inputs to outputs is computable. As a consequence of the continuity theorem, simple real functions like the step function $s(x) :=$ (0 if $x < 0$, 1 otherwise) and the Gauß staircase $g(x) := \lfloor x \rfloor$ (the integer part of x) are not computable (Fig 1.3). Obviously, these functions are *easily definable* in our mathematical language, but "easily definable" does not mean "computable" in general. As far as we know neither the step function or the Gauß staircase nor any other non-continuous real function can be computed by physical devices.

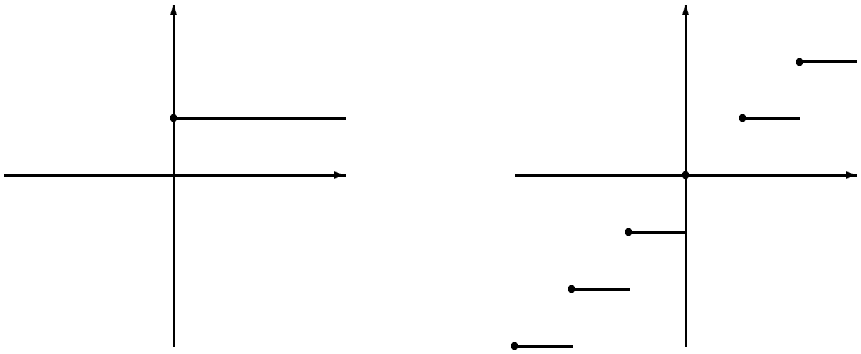

Fig. 1.3. Two non-computable real functions

Some people do not accept the model of computation presented here, since they believe that the above simple functions are or should be called computable. Although the Gauß staircase is not computable according to our present definition, we will see that the framework of TTE admits to define other natural computability concepts such that it is a computable object (Exercise 4.3.2).

1.3.4 Subsets of Real Numbers

A subset $A \subseteq \mathbb{N}$ is *decidable* or *recursive*, iff its characteristic function $\mathrm{cf}_A : \mathbb{N} \to \mathbb{N}$ is computable. Assume for the moment that we call a subset $A \subseteq \mathbb{R}$ recursive, iff its characteristic function $\mathrm{cf}_A : \mathbb{R} \to \mathbb{R}$ is computable. Then $\mathrm{cf}_A : \mathbb{R} \to \mathbb{R}$ is not continuous and hence not computable by the continuity theorem, unless $A = \emptyset$ or $A = \mathbb{R}$. Therefore, this definition is useless.

The *distance function* $d_A : \mathbb{R}^n \to \mathbb{R}$, defined by $d_A(x) := \inf_{y \in A} |y - x|$ is a more useful generalization of the discrete characteristic function. Fig. 1.4 shows the function $1 - \mathrm{cf}_{[0;1]}$ and its continuous counterpart $d_{[0;1]}$. We call a (closed) subset $A \subseteq \mathbb{R}^n$ *recursive*, iff its distance function $d_A : \mathbb{R}^n \to \mathbb{R}$ is computable. It turns out that simple sets like intervals $[a;b] \subseteq \mathbb{R}$ with computable endpoints a and b, the set $\{(x,y) \in \mathbb{R}^2 \mid x \leq y\}$ and the graph $\{(x, f(x)) \mid x \in \mathbb{R}\}$ of any computable function $f : \mathbb{R} \to \mathbb{R}$ are recursive.

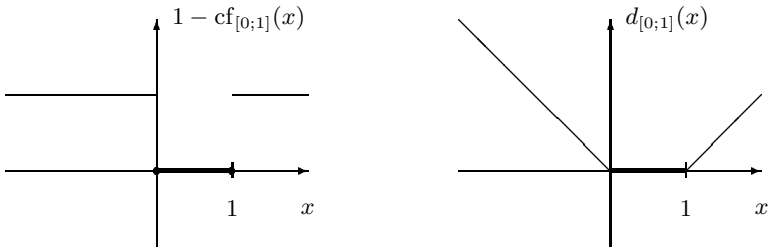

Fig. 1.4. $1 - \mathrm{cf}_{[0;1]}$ and $d_{[0;1]}$

For defining computability on subsets of $\mathbb{R}$, we introduce naming systems of sets of subsets. Since the set $2^{\mathbb{R}}$ of all subsets of $\mathbb{R}$ is too large, we consider only subsets of $2^{\mathbb{R}}$. Here we merely introduce a naming system for the set $\mathcal{O}(\mathbb{R})$ of the open subsets of real numbers. Later we will study various naming systems of the open, the closed and the compact subsets of $\mathbb{R}^n$. As is well known, a subset $U \subseteq \mathbb{R}$ is *open*, iff it is the union of a set of open intervals with rational endpoints.

We define: a *name* of an open subset $U \subseteq \mathbb{R}$ is a sequence $(I_0, I_1, \ldots)$ of open intervals with rational boundaries such that $U = I_0 \cup I_1 \cup \ldots$. In the general case $U \subseteq \mathbb{R}^n$, each I_k is a product of n open intervals with rational endpoints. We call an open set recursively enumerable, iff it has a computable name.

Examples of recursively enumerable open sets are the open intervals $(a; b)$ with computable endpoints a and b, the set $\{(x, y) \subseteq \mathbb{R}^2 \mid x < y\}$ and every interval $(0; x_A)$ where x_A is from Example 1.3.2 with r.e. (recursively enumerable) but non-recursive $A \subseteq \mathbb{N}$. As for real functions, we call a function on the set of open sets computable, iff some machine maps names of arguments to names of the results. The functions *intersection* and *union* on open sets are computable. For example. let M be a machine with two input files which transforms any two names $(I_0, I_1, \ldots)$ and $(J_0, J_1, \ldots)$ of open sets to a sequence $(K_0, K_1, \ldots)$ which is a list of all $I_m \cap J_n$ with $m, n \in \mathbb{N}$. Then M computes the intersection on $\mathcal{O}(\mathbb{R})$.

We define computability for functions of mixed type accordingly. For example, the function $f : \mathbb{R} \to \mathcal{O}(\mathbb{R})$ with $f(x) := \mathbb{R} \setminus \{x\}$ is computable. A machine computing f maps any name of $x \in \mathbb{R}$ to a name of $\mathbb{R} \setminus \{x\}$. Since a computable function maps computable names to computable names, $X \cap Y$ and $X \cup Y$ are recursively enumerable open sets, if X and Y are recursively enumerable open sets, and $\mathbb{R} \setminus \{x\}$ is a recursively enumerable open set, if x is a computable real number.

1.3.5 The Space C[0; 1] of Continuous Functions

As a last example we consider the set C[0; 1] of continuous functions $f : [0; 1] \to \mathbb{R}$. First, we define a naming system. Let us call a function $f : [0; 1] \to \mathbb{R}$ a *rational polygon*, iff its graph is a polygon with finitely many vertices with rational coordinates. We call the set $B(p, r) = \{f \in \mathrm{C}[0; 1] \mid d(f, p) < r\}$, where $d(f, p) = \max_{x \in [0;1]} |f(x) - p(x)|$ the *function ball* with *center* $p \in \mathrm{C}[0; 1]$ and *radius* $r > 0$. In this section, we generalize our naming system for real numbers as follows. A name of a continuous function $f : [0; 1] \to \mathbb{R}$ is a sequence $(B_0, B_1, \ldots)$, where B_n is a function ball of radius 2^{-n} with $f \in B_n$ the center of which is a rational polygon. Fig. 1.5 shows such a ball.

A name $(B_0, B_1, \ldots)$ of f encloses f arbitrarily narrowly. A function $f : [0; 1] \to \mathbb{R}$ can have a computable name, and it can be computable (by a machine transforming each name of real numbers $x \in [0; 1]$ to a name of $f(x)$). Later, we will prove that these properties are equivalent.

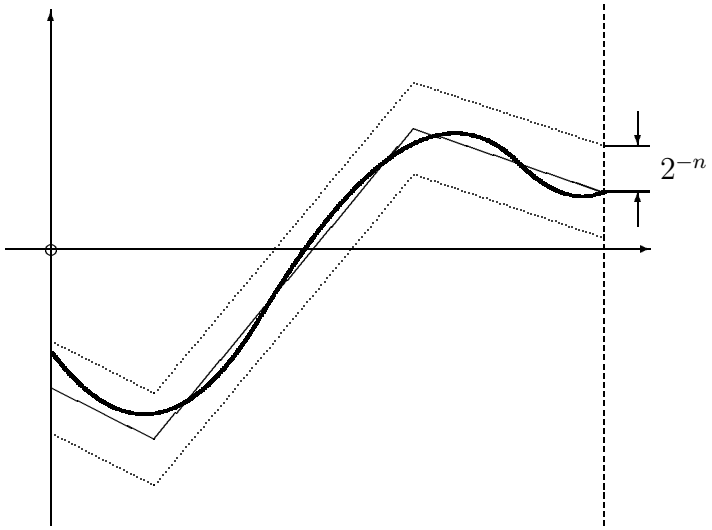

Fig. 1.5. A function f in a function ball with radius 2^{-n} with a rational polygon as center and radius 2^{-n}

Our naming systems of $\mathbb{R}$ and $\mathrm{C}[0;1]$ make the evaluation function $\mathrm{Apply} : \mathrm{C}[0;1] \times [0;1] \to \mathbb{R}$, defined by $\mathrm{Apply}(f,x) := f(x)$, computable, that is, there is a machine which transforms any name of any f and any name of any x to a name of $f(x)$. There are many other naming systems of $\mathrm{C}[0;1]$, for which the evaluation function becomes computable, however, among all of these our naming system is the "weakest" one (we will explain this later). Thus, our naming system for $\mathrm{C}[0;1]$ is tailor-made for the evaluation function.

Integration $f \mapsto \int_0^1 f(x)\mathrm{d}x$ on $\mathrm{C}[0;1]$ is an important computable operator, while differentiation $f \mapsto f'$ for continuously differentiable $f \in \mathrm{C}[0;1]$ is not computable. By the classical *intermediate value theorem* every continuous function $f : [0;1] \to \mathbb{R}$ with $f(0) < 0 < f(1)$ has a zero, that is, a point x with $f(x) = 0$. Unfortunately, there is no computable operator for zero-finding working correctly for all continuous functions $f : [0;1] \to \mathbb{R}$ with $f(0) < 0 < f(1)$. However, a restricted problem has a computable solution: the operator $Z : f \mapsto$ (the zero of f) for continuous increasing functions with $f(0) < 0 < f(1)$ is computable. Later, we will discuss in detail the problem of zero-finding under various assumptions.

1.3.6 Computational Complexity of Real Functions

Ordinary discrete complexity theory studies resources like time or storage used by machines for computing functions [HU79]. If M is a Turing machine computing a word function $f : \Sigma^* \to \Sigma^*$, then complexity theory considers

$\mathrm{Time}_M(x)$, the number of steps which M makes on input x before halting. This concept cannot be transferred to machines which map infinite sequences to infinite sequences, since these machines do not halt. However, we can consider for each input sequence $s = (I_0, I_1, \ldots)$ and for each number n the number $\mathrm{Time}_M(s)(n)$ of steps which M makes before it prints the first interval J_k of its output $(J_0, J_1, \ldots)$ of length $< 2^{-n}$. For a machine M computing a real function $f : \mathbb{R} \to \mathbb{R}$ we would like to have a complexity $\mathrm{Time}_M(x)(n)$, which depends on the real number x but not on a name of x. For this pupose we define

$$\mathrm{Time}_M(x)(n) := \max\{\mathrm{Time}_M(s)(n) \mid s \text{ is a name of } x\} .$$

Since in our naming system we admit arbitrarily redundant names of the real numbers, this maximum does not exist.

However, for the *signed digit representation* of the real numbers, we obtain very natural complexity results. In the signed digit representation, a name of a real number is an infinite binary fraction where the digit $\overline{\mathtt{1}} := -1$ can be used in addition to the digits $\mathtt{0}$ and $\mathtt{1}$, that is, $a_n \ldots a_0 \bullet a_{-1} a_{-2} \ldots$ with $a_k \in \{\mathtt{0}, \mathtt{1}, \overline{\mathtt{1}}\}$ is a name of $x := \sum_{k=n}^{-\infty} a_k$. These names encode strongly normalized sequences of nested intervals and induce the same computability on $\mathbb{R}$ as our standard names. The induced computational complexity of real functions is sometimes called "bit complexity". Bit complexity has been studied by several authors, in particular, by Ko [Ko91, Ko98], who applied concepts from discrete complexity theory to prove upper and lower bounds for the complexity of basic numerical operations. It turns out that Turing machines operating on signed digit representations of real numbers can compute addition in time $O(n)$ and multiplication, sin, exp and log in time $O(n^2)$.

1.4 Prerequisites and Notation

We assume that the reader is familiar with the basic concepts of analysis, that is, the theory of real numbers and functions, and of ordinary computability theory at undergraduate level in computer science. For details the reader should consult standard textbooks, e.g. parts of [Die60, HU79].

By $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$ we denote the set $\{0, 1, \ldots\}$ of natural numbers, of integer numbers, of rational numbers and of real numbers, respectively. For a set A the power set of A, that is, the set of all subsets of A, is denoted by 2^A and the set of all finite subsets of A is denoted by $\mathrm{E}(A)$. If $X = A_1 \times \ldots \times A_n$ is a Cartesian product of sets, then for $n = 0$, $X = \{()\}$ is the set the only element of which is the *empty tuple* "()" .

If $\leq$ is a binary relation on $A \times B$, $x < y$ will abbreviate "$x \leq y$ and $y \not\leq x$".

We will consider multi-valued partial functions and as special cases partial functions and total functions. A *correspondence* or *multi-valued partial*

function from A to B is a triple $f = (A, B, \mathrm{R}_f)$ such that $\mathrm{R}_f \subseteq A \times B$, where A is called the *source*, B the *target* and R_f the *graph* of f. For $X \subseteq A$ we define the *image* of X under f by

$$f[X] := \{b \in B \mid (\exists a \in X)(a, b) \in \mathrm{R}_f\} .$$

We define the *inverse*, the *domain* and the *range* of f by

$$f^{-1} := (B, A, \mathrm{R}_f^{-1}) \text{ such that } \mathrm{R}_f^{-1} := \{(b, a) \mid (a, b) \in \mathrm{R}_f\} ,$$
$$\mathrm{range}(f) := f[A] \quad \text{and}$$
$$\mathrm{dom}(f) := f^{-1}[B] ,$$

respectively. The multi-valued partial function f is completely defined by the family of sets $f[\{a\}]$ $(a \in A)$.

Usually we will denote a correspondence f from A to B by $f :\subseteq A \rightrightarrows B$. A *partial function* $f :\subseteq A \to B$ from A to B is a multi-valued function $f :\subseteq A \rightrightarrows B$ such that the set $f[\{a\}]$ contains only one element for each $a \in \mathrm{dom}(f)$. A *total function* $f : A \to B$ from A to B is a partial function $f :\subseteq A \to B$ such that $\mathrm{dom}(f) = A$. The set of all (total) functions $f : A \to B$ is denoted by B^A. For a partial function $f :\subseteq A \to B$, $f(a)$ denotes the single element from $f[\{a\}]$, if $a \in \mathrm{dom}(f)$, and we will write $f(a) = \mathrm{div}$, if $a \notin \mathrm{dom}(f)$. Usually we will call f a *function*, if f is a partial function or a total function.

For multi-valued functions $f_i :\subseteq A \rightrightarrows B_i$ $(i{-}1 \ldots, k)$ define $(f_1, \ldots, f_k) :\subseteq A \rightrightarrows B_1 \times \ldots \times B_k$ by $(f_1, \ldots, f_k)[\{a\}] := f_1[\{a\}] \times \ldots \times f_k[\{a\}]$. For multi-valued functions $f :\subseteq A \rightrightarrows B$ and $g :\subseteq B \rightrightarrows C$ define the *composition* $g \circ f :\subseteq A \rightrightarrows C$ by $a \in \mathrm{dom}(g \circ f)$, iff $a \in \mathrm{dom}(f)$ and $f[\{a\}] \subseteq \mathrm{dom}(B)$, and $g \circ f[\{a\}] := g[f[\{a\}]]$ for all $a \in \mathrm{dom}(g \circ f)$.

For a multi-valued function $f :\subseteq A \to B$, $X \subseteq A$ and $Y \subseteq B$, we define the restrictions in the source, the target, the domain and the range as follows:

$$\begin{array}{lll}
\text{source:} & f|_X :\subseteq X \to B, & f|_X := (X, B, \mathrm{R}_f \cap X \times B) \\
\text{domain:} & f\rfloor_X :\subseteq A \to B, & f\rfloor_X := (A, B, \mathrm{R}_f \cap X \times B) \\
\text{target:} & f|^Y :\subseteq A \to Y, & f|^Y := (A, Y, \mathrm{R}_f \cap A \times Y) \\
\text{range:} & f\rceil^Y :\subseteq A \to B, & f\rceil^Y := (A, B, \mathrm{R}_f \cap A \times Y) .
\end{array}$$

For combinations of restrictions we use the symbols $|, \rceil, \rfloor$ and $]$, for example $f]_X^Y$. Usually, $\mathrm{id}_X : X \to X$ denotes the *identity function* on X.

For any set Σ, Σ^n denotes the set of all words over Σ of length n, $\Sigma^{\leq n}$ the set $\Sigma^0 \cup \ldots \cup \Sigma^n$ and Σ^* the set of all finite words over Σ. We denote the *length* of a word w by $|w|$ and the *empty word* which has length 0 by λ. By Σ^ω we denote the set $\{p \mid p : \mathbb{N} \to \Sigma\} = \{a_0 a_1 a_2 \ldots \mid a_i \in \Sigma\}$ of all infinite sequences over Σ. Occasionally finite or infinite sequences of symbols will be called *strings*.

We extend concatenation from $\Sigma^* \times \Sigma^*$ to $\Sigma^* \times \Sigma^\omega$. Consider $u, v, w \in \Sigma^*$ and $p \in \Sigma^\omega$. If $x = uvw \in \Sigma^*$ and $q = uvp \in \Sigma^\omega$, then u is a *prefix* of x

and q (abbreviated by $u \sqsubseteq x$ and $u \sqsubseteq q$, respectively), v is a *subword* of x and q (abbreviated by $v \lhd x$ and $v \lhd q$, respectively) and w is a *suffix* of x. A set $A \subseteq \Sigma^*$ is called *prefix-free*, iff $x \not\sqsubseteq y$ for all $x, y \in A$ with $x \neq y$. We define $u^n := uu \ldots u$ (n times) as usual and $u^\omega := uuu \ldots \in \Sigma^\omega$. By $p_{<n}$ we denote the prefix $p(0) \ldots p(n-1)$ of length n of p and by $p_{\leq n}$ the prefix $p(0) \ldots p(n)$. We extend concatenation straightforwardly to sets of finite or infinite sequences: $AB := \{up \mid u \in A, p \in B\}$ for $A \subseteq \Sigma^*$ and $B \subseteq \Sigma^*$ or $B \subseteq \Sigma^\omega$. In particular, $A^n := A \ldots A$ (n factors). (If $A \subseteq \Sigma^*$, it will be clear from the context, whether A^n denotes the Cartesian product or the concatenation product.) The prefix relation can be expressed by means of product: $u \sqsubseteq v \iff v \in u\Sigma^*$ (for $u, v \in \Sigma^*$) and $u \sqsubseteq q \iff q \in u\Sigma^\omega$ (for $u \in \Sigma^*$ and $p \in \Sigma^\omega$).

Later Σ will be a non-empty finite set, an *alphabet*, the symbols of which we will denote in typewriter style (`abc...012...#.$` etc.). In emphasizing that an expression denotes a word, we occasionally set it into quotes. For example: if u and v denote words and if `0` and `;` are elements of Σ, then "u`0;`v" denotes the word which begins with u, has `0` and `;` as the next symbols and ends with the word v.

Ordinary "Type-1" recursion theory considers the computable number functions $f :\subseteq \mathbb{N}^k \to \mathbb{N}$ and the computable word functions $g :\subseteq (\Sigma^*)^k \to \Sigma^*$. Computability can be transferred to other sets M by means of numberings. A *numbering* of a set M is a surjective (that is, "onto") function $\nu :\subseteq \mathbb{N} \to M$. For a given finite alphabet $\Sigma = \{a_1, \ldots, a_n\}$ define the bijective standard numbering $\nu_\Sigma : \mathbb{N} \to \Sigma^*$ of Σ^* as follows:

$$\nu_\Sigma^{-1}(\lambda) := 0;\ \nu_\Sigma^{-1}(a_{i_k} \ldots a_{i_0}) := i_k \cdot n^k + \ldots + i_0 \cdot n^0 \ .$$

Then $f :\subseteq \mathbb{N} \to \mathbb{N}$ is computable, iff $\nu_\Sigma \circ f \circ \nu_\Sigma^{-1} :\subseteq \Sigma^* \to \Sigma^*$ is computable (correspondingly for $f :\subseteq \mathbb{N}^m \to \mathbb{N}$). Occasionally, we will extend the definition of computable functions to functions of mixed type ($\mathbb{N}$ and Σ^*) by calling ν_Σ and ν_Σ^{-1} computable and closing under composition. This way we can, for example, define the r.e. subsets $A \subseteq \Sigma^* \times \mathbb{N} \times \Sigma^*$ and the computable functions $f :\subseteq \mathbb{N} \times \Sigma^* \to \mathbb{N}$.

We will need the bijective Cantor pairing function $\langle\ ,\ \rangle : \mathbb{N}^2 \to \mathbb{N}$, defined by $\langle x, y\rangle := y + (x+y)(x+y+1)/2$, which is computable, as well as the projections $\langle\ \rangle_1$ and $\langle\ \rangle_2$ of its inverse. For $n > 2$ arguments, we define inductively

$$\langle x_1, \ldots, x_n\rangle := \langle\langle x_1, \ldots, x_{n-1}\rangle, x_n\rangle$$

as usual.