A Refined Model of Computation for Continuous Problems

Klaus Weihrauch *

FernUniversität Hagen, D-58084 Hagen, Germany

Received March 27, 1997

The real-number model of computation is used in computational geometry, in the approach suggested by Blum, Shub, and Smale and in information based complexity. In this paper we propose a refinement of this model, the TTE-model of computation. In contrast to the real-number model, which is unrealistic or at least too optimistic, the TTE-model is very realistic; i.e., for TTE-algorithms there are digital computers, which behave exactly the same way as predicted by the theoretical model. We start with a detailed discussion of some objections to the real-number model. We introduce the refined model by adding the condition “every partial input or output information of an algorithm is finite” to the assumptions of the IBC-model of computation. First, we explain computability and computational complexity in TTE for the simple case of real functions. Then we apply the refined model to two typical IBC-problems, integration and zero-finding on the space $C[0; 1]$ of continuous real functions. We study computability and computational complexity and compare TTE-results with IBC-results. Finally, we briefly discuss the computation of discontinuous problems. This paper does not present new technical results but should be considered as a fresh impetus to reflect on models of computation for numerical analysis and as an invitation to try out the TTE-model of computation in information based complexity. © 1998 Academic Press

1. INTRODUCTION

Information-based complexity (IBC) typically studies the computational complexity of infinite dimensional problems. Usually, such a problem is a function $S: F \to G$, where $F$ is a subset of a linear space and $G$ is a normed linear space. We wish to have an algorithm which for any $\varepsilon > 0$ and $f \in F$ determines an $\varepsilon$-approximation to $S(f)$. For precise formulations and comprehensive presentations we refer the reader to the work by Traub, Wasilkowski, and Woźniakowski [TW80, TWW88] and [TW91, TW93], which are more recent reviews.

*E-mail: Klaus.Weihrauch@Fernuni-Hagen.de.
The computational model considered in IBC can be described as real RAMs, real random access machines, with oracle calls. A precise definition has been given by Novak [Nov95]. Roughly speaking, a real RAM is a flowchart program which manipulates real numbers, where each real number is handled as an entity. It is assumed that addition, multiplication, comparison, and some other basic operations on real numbers can be performed in one step with unit cost. Real RAMs cannot handle the “abstract” objects \( f \in F \), in general. Therefore, in the IBC computational model pieces \( x \in \mathbb{R} \) of partial information about the input \( f \in F \) can be obtained on demand by “oracle calls” during the computation, where each oracle call has constant cost \( c, c > 0 \). The output of a computation is a vector of real numbers, approximating the correct value \( S(f) \) with error \( \varepsilon \), where \( \varepsilon \) may be considered as a fixed parameter or as an input variable (uniform case).

Traub and Woźniakowski [TW91] justify the use of the real-number model as follows:

The rationale for the real-number model is that fixed-precision floating-point numbers are almost universally used for numerical calculations whether they occur in science, engineering, or economics. The cost of floating-point operations is independent of the size of operands. Complexity results are essentially the same as in the real-number model provided optimal algorithms are stable. The real-number model is used in IBC to decouple complexity from round-off issues. The numerical stability of optimal algorithms requires further study; ...
These statements hold also for IBC-algorithms. In the following we discuss them in detail.

(a) It is generally accepted that Turing machines can be realized by digital computers [HU79]. A physical realization of a Turing machine can be described very roughly as a finite device (control) with a potentially infinite tape as its memory which reads a finite word \( x \in \Sigma^* \) as an input to its memory, operates for a while, and then writes a finite word \( y \in \Sigma^* \) as the output. The physical realization behaves in exactly the same way as predicted from the mathematical model.

Real RAMs may be considered as mathematical generalizations of Turing machines obtained by substituting the set of real numbers \( \mathbb{R} \) for the finite (!) alphabet \( \Sigma \). Often it is assumed tacitly that also real RAMs can be realized accordingly by physical devices. Such a device must be able to read and to write an arbitrary real number in a finite amount of time, and it must be able to store an arbitrary real number. In fact, classical physics, i.e., Newton’s mechanics and Maxwell’s electrodynamics, admits analog devices which can read, store, or write arbitrary real numbers. A real number could be realized, e.g., by a voltage or an amplitude of a mechanical wave.

However, according to all experience from physics and electrical engineering, no such devices exist in reality. Every real information channel has only finite capacity; therefore, in any finite amount of time only finitely many bits of information can be transferred. Since there are uncountably many real numbers, in general, infinitely many bits of information are necessary to identify a single number; hence no channel can transfer an arbitrary real number in a finite amount of time. Similarly, every physical storage or memory is finite. Even if we supply bigger and bigger finite extensions on demand, at most countably many different objects can be distinguished. Therefore, no physical memory can store arbitrary real numbers. These limitations for channels and memories can be explained quite satisfactorily by quantum mechanics and thermodynamics. They must be accepted as facts. Therefore, neither real RAMs nor IBC-algorithms can be realized correctly by physical devices. As an example consider the step function \( s : \mathbb{R} \to \mathbb{R}, s(x) := (0 \text{ if } x < 0, 1 \text{ otherwise}) \). Obviously, this function is real RAM-computable, but we cannot imagine any physical device which realizes it correctly.

(b) Physicists, engineers, and other scientists usually write their numerical algorithms in a programming language like PASCAL or FORTRAN. Such languages have a data type called “real.” Two semantics can be associated with any program: mathematical semantics and computer semantics. In mathematical semantics, actually real numbers are assigned to the variables of type “real.” It is assumed that all operations on objects of type “real” like addition, multiplication, and comparison are performed mathematically correctly. With this mathematical semantics a program may be considered (essentially) as a real RAM. In computer semantics, floating-point numbers are assigned to the variables of type “real,” where the set \( FN \) of floating-point numbers may depend on the computer. A
typical set is $FN := \{a \cdot 10^{b-10} \mid a, b \in \mathbb{Z}, -10^{10} < a < 10^{10}, -100 < b < 100\}$ (10 decimal digits for the mantissa and 2 for the exponent). Notice that $FN$ is a finite set which is very dense in the neighborhood of 0 and sparse far from 0. Since the set $FN$ as a subset of the real numbers is not closed under ordinary addition, in computer semantics addition on type “real” means ordinary addition followed by a roundoff of the result, which is usually not a floating-point number, to a floating-point number closest to it. This remark holds accordingly for the other available operations with values of type “real.” By $f : \subseteq X \rightarrow Y$ we shall denote a function from $X$ to $Y$ the domain of which is a subset of $X$.

Consider now for simplicity programs with a single input and a single output each of type real. Then the mathematical semantics of a program $P$ is a function $P_{\text{math}} : \subseteq \mathbb{R} \rightarrow \mathbb{R}$, while the computer semantics is a function $P_c : \subseteq FN \rightarrow FN$ (where $FN$ depends on the computer). Let $P'_{\text{math}} : \subseteq FN \rightarrow \mathbb{R}$ be the restriction of $P_{\text{math}}$ to the set $FN$ of floating-point numbers. We discuss the relation between the functions $P_{\text{math}}, P'_{\text{math}},$ and $P_c$. A scientist usually writes a program $P$ which is correct w.r.t. the mathematical semantics. Since his computer can read only inputs from the set $FN$ of floating-point numbers, at most the finite part $P'_{\text{math}}$ of $P_{\text{math}}$ can be actually computed. For a smooth (i.e., continuous slowly varying) function $P_{\text{math}}, P'_{\text{math}}$ may be a reasonable “approximation” to $P_{\text{math}}$. Because of round-off errors, the function $P_c$ realized by the computer differs from $P'_{\text{math}}$. Again for a smooth function $P_{\text{math}}$ and for short computations, $P_c$ may be a reasonable “approximation” to $P'_{\text{math}}$, since in this case roundoff errors have only a small influence on the results.

For reducing the influence of roundoffs, programming languages supply “multiple precision” floating-point numbers with mantissas (and possibly exponents) of multiple length. Let $FN_k$ be the set of floating-point numbers obtained from $FN$ by increasing the lengths of the mantissas and exponents by a factor $k$. Let $P_{ck}$ be the computer semantics based on $FN_k$. If $P_{\text{math}}$ is continuous, then for reasonable programs the sequence $P_{c1}, P_{c2}, \ldots$ “converges” to an extension of $P'_{\text{math}}$ in a reasonable way. For discontinuous functions, however, such approximations are not possible in general. As an example consider the function $f : \subseteq \mathbb{R} \rightarrow \mathbb{R}$ defined by

$$f(x) := \begin{cases} 1, & \text{if } x \in \mathbb{Q} \\ 0, & \text{if } x \notin \mathbb{Q}; \ x^2 \in \mathbb{Q} \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

The function $f$ is real RAM-computable; i.e., there is a real RAM program $P$ with $f = P_{\text{math}}$. Since $P'_{\text{math}}$ is constantly 1 for all inputs $x \in FN$, it is not a reasonable approximation to $f$. Furthermore, the sequence $P_{c1}, P_{c2}, \ldots$ does not converge to $f$ in a reasonable way, since $f(x) = 1$ for all $k$ and $x \in FN_k$.

(c) Consider the functions $P_{\text{math}}$ and $P_{ck}$ from (b). If $P_{\text{math}}$ is continuous, then the sequence $P_{c1}, P_{c2}, \ldots$ “converges” to $P_{\text{math}}$. If, however, $P_{\text{math}}$ has only a fast increasing modulus of continuity then the convergence may be very slow. For reducing the influence of roundoff errors, floating-point numbers with
very long mantissas have to be used ($P_{ck}$ with large $k$). But handling long floating-point numbers on a computer is more expensive than handling short ones. Therefore, in many cases the assumption of unit cost for a basic operation on variables of Type “real” becomes extremely unrealistic, if the required error bound for the result approaches 0.

(d) In programming languages as well as models of computation, usually programs for functions $f: X \rightarrow Y$ and $g: Y \rightarrow Z$ can be combined easily to a program for the composition $gf: X \rightarrow Z$ by using the output of the first program as the input for the second one. IBC-algorithms do not have this property, in general. As an example, consider the exponential function $\exp: \mathbb{R} \rightarrow \mathbb{R}$. It cannot be computed exactly on a real RAM. As a substitute there is a real RAM $M$ which for any $x \in \mathbb{R}$ and $\varepsilon > 0$ as inputs determines some $y \in \mathbb{R}$ with $|y - \exp(x)| < \varepsilon$. However, the results of this machine $M$ cannot be used as inputs for a second version of $M$ (for computing $\exp(\exp(x))$), since the latter requires exact inputs and cannot use approximate ones. Correspondingly, in general IBC-algorithms cannot be combined for computing the composition, since the outputs of a first algorithm and the inputs of a second algorithm generally are of different types.

The discussions in (b) and (c) are based on physical realizations defined by the floating-point semantics ($P_c$, $P_{ck}$). The results remain essentially the same, however, if other realizations are considered, e.g., with dynamically extendible floating-point numbers, rational numbers, or algebraic numbers.

All these considerations show that algorithms formulated in the real RAM-model or the IBC-model derived from it may be very unrealistic or at least too optimistic. Properties like correctness or computational complexity of these mathematical algorithms are not shared by their physical realizations, in general. Additional considerations are necessary for distinguishing the realistic results, i.e., those which describe properties of realizing computations correctly and which are of practical relevance, from unrealistic ones. In IBC, lower complexity bounds are realistic, and usually the good “condition” of the problem and “stability” of the algorithm guarantee correct realization with the predicted complexity.

Since it would be very unsatisfactory to decide whether a result is realistic or not, merely by performing computer experiments, we need a model for computable analysis which describes the behavior of computers adequately. In this contribution we propose such a realistic model of computation for IBC. We can obtain it by a slight, but significant, modification of the IBC-computational model. We agree with the basic assumption of IBC that information about input and output objects is partial. We avoid, however, the unrealistic assumption that real numbers can be read, stored, or written completely as entities by the following condition:

*Every partial input or output information is finite.*
In the formal definition we shall have a countable supply $\sigma$ of pieces of information, each of which has a finite name given by a notation $\nu : \subseteq \Sigma^* \to \sigma$. The elements of $\sigma$ will be called “atomic properties.” In the new model no longer arbitrary real numbers can be used as inputs or outputs, but only rational numbers, intervals with rational boundaries, or maybe algebraic numbers. Actually, the computer or machine reads finite names and writes finite names of such objects, and there is no necessity to store or to perform operations on arbitrary real numbers during a computation. In particular, branchings like “$x \leq 0$” for real numbers $x$, which induce discontinuity and instability and cause severe problems in numerical realizations, are no longer needed.

Consider an IBC problem $S : F \to G$. A solution could be given by a machine or device, which for any atomic property $I$ of some $f \in F$ produces an atomic property $J$ of $S(f)$. Such a machine may be extended to a machine which obtains an infinite sequence $I_0, I_1, \ldots$ of atomic properties of some $f \in F$ as input and produces an infinite sequence $J_0, J_1, \ldots$ of atomic properties of $S(f)$. In the mathematical model the machine works forever, in general; in practice it will work only until the user stops it or some storage or time bound is reached. Usually, an input $I_0, I_1, \ldots$ for a machine $M$ will be a sequence of atomic properties identifying an element $f \in F$, and the output $J_0, J_1, \ldots$ will be a sequence of atomic properties identifying $S(f)$.

Notice, that in this model the input and the output are of the same type; hence, machines can be combined for computing the composition of functions (see (d) above). Since the active device, the machine $M$, gets finite words as inputs and produces finite words as outputs, there is no need for a new computational model (like the real-number model). For a mathematical foundation we consider Turing machines, which transform infinite sequences of words to infinite sequences of words. For more convenient programming, we may use programs from some higher level programming language (of course without using the type “real”) instead of Turing machines. As a consequence, none of the problems discussed in (a) and (b) above occur for this model. Furthermore, we do not need a new ad hoc definition of computational complexity (c), since we have time and tape complexity as the generally accepted realistic complexity measures for Turing machines.

A slight modification adjusts the above machine $M$ to the IBC-model of computation. Assume, there is a metric $d$ on the space $G$. Let $M_1$ be a machine which gets some $n \in \omega$ as an additional input and then operates like $M$ until some $J_k$ with $d(S(f), J_k) < 2^{-n}$ is obtained. During the finite computation only finitely many properties $I_0, I_1, \ldots, I_m$ can be used. Reading some information $I_j$ corresponds to an oracle call in the IBC-model.

The modified computational model suggested here as a refined more realistic model for IBC is based on a definition of computable real functions by Grzegorczyk [Grz55, Grz57]. Its theory TTE (“Type 2 Theory of Effectivity”) has been developed already for a number of years. The associated complexity is
often called “bit-complexity.” For references and more information the reader is referred to recent reviews [Wei95b, Wei95a, Wei97] containing several examples and mathematical framework and to Ko’s book on the computational complexity of real functions [Ko91]. Also Pour-El’s [PER88] concept of computability is consistent with this model.

2. COMPUTATION OF REAL FUNCTIONS

For explaining the model in this section we consider “problems” $S : F \to G$, where $F \subseteq \mathbb{R}^n$ and $G = \mathbb{R}$. According to the new concept we need a countable set $\sigma$ of “finite” properties of real numbers. Although we are completely free, we choose for $\sigma$ a set of properties, which is mostly used, if real numbers are handled, the set of open intervals with rational boundaries:

$$\sigma := \{(r; s) \mid r, s \in \mathbb{Q}, r < s\}.$$

First, we discuss only computability and not computational complexity. Therefore, we do not fix an encoding of the elements of $\sigma$ by words, but we assume for simplicity that our machines can read and write rational numbers and can perform addition, multiplication, comparison, etc. on rational numbers. The following definition of a computable real function is equivalent to Grzegorczyk’s original one [Grz55]:

A real function $S : \subseteq \mathbb{R} \to \mathbb{R}$ is called computable, iff there is a machine $M$ with the following property. If $M$ obtains as input any list $I_0, I_1, \ldots$ of all (!) properties $I \in \sigma$ of some $x \in \text{dom}(S)$ (in whatever order), then it produces a list of all (!) properties $J \in \sigma$ with $f(x) \in J$.

The strong conditions “all” can be weakened considerably. An easy proof shows, that in the case of real numbers it is equivalent to consider for inputs and outputs sequences $K_0, K_1, \ldots$ of open intervals from $\sigma$, which converge to some $x \in \mathbb{R}$ (i.e., $K_0 \supseteq K_1 \supseteq \cdots$ and $\{x\} = \cap K_i$). The following examples illustrate the definition of computable real functions.

Consider the function $S : \mathbb{R} \to \mathbb{R}$, $S(x) := 3x$. We show that there is a machine $M$, which transforms any sequence of intervals $I_0, I_1, \ldots$ from $\sigma$ to a sequence of intervals $J_0, J_1, \ldots$ from $\sigma$ such that for all $x \in \mathbb{R}$:

$I_0, I_1, \ldots$ converges to $x \implies J_0, J_1, \ldots$ converges to $3x$.

Define $q : \sigma \to \sigma$ by $q(r; s) := (3r; 3s)$. There is a machine $M$, which reads the input $I_0, I_1$ successively and writes $J_0 := q(I_0), J_1 := q(I_1), \ldots$ on the output. Obviously, if the input $I_0, I_1, \ldots$ converges to $x$ then the output $J_0, J_1, \ldots$ converges to $3x$.

As a further example, consider the function $S : \subseteq \mathbb{R} \to \mathbb{R}$, $S(x) := (\sqrt{x}$ if $x \geq 0$, undefined otherwise). In this case we cannot define a function $q : \sigma \to \sigma$
with \( \{S(x) \mid x \in I\} = q(I) \) for \( I \in \sigma \) as in the first example. It suffices, however, to have \( \{S(x) \mid x \in I\} \subseteq q(I) \), where \( q(I) \) is not too big. Let \( q: \sigma \to \sigma \) be a function such that for all \( 0 \leq a < b \) the interval \( (c; d) := q(a; b) \) satisfies

\[
a - (b - a) < c^2 \leq a < b < d^2 < b + (b - a).
\]

Since the rational squares are dense in \( \mathbb{R} \), some machine can compute appropriate numbers \( c, d \) from \( a, b \) by an exhaustive search. Now, let \( M \) be a machine which successively reads the input intervals \( I_0, I_1, \ldots \) and for \( (a_n; b_n) := I_n \) writes \( q(a_n; b_n) \) if \( 0 \leq a_n < b_n \), writes \( q(0; b_n) \) if \( a_n < 0 < b_n \) and writes nothing if \( b_n \leq 0 \). If the input \( I_0, I_1, \ldots \) converges to \( x \geq 0 \) then the output \( J_0, J_1, \ldots \) converges to \( \sqrt{x} \). Otherwise, the machine computes forever with some finite sequence \( J_0, J_1, \ldots, J_k \) as the output.

In a similar way, computability of real functions like \( x + y, x \cdot y, x/y, e^x, \sin(x), \max(x, y), \log(x), \ldots \) can be shown [Wei95a]. All these functions are continuous. More generally, the following theorem holds.

*Every Grzegorczyk computable real function is continuous.*

Because of its importance we sketch a proof. For simplicity, consider a computable function \( S: \subseteq \mathbb{R} \to \mathbb{R} \). Then there is some machine computing it. Let \( S(x) \in U \) for some \( x \in \mathbb{R} \) and some open set \( U \subseteq \mathbb{R} \). There is some sequence \( I_0, I_1, \ldots \) of intervals from \( \sigma \) converging to \( x \). By assumption, the machine transforms it to a sequence \( J_0, J_1, \ldots \) of intervals converging to \( S(x) \). Then \( S(x) \in J_n \subseteq U \) for some \( n \). For producing \( J_n \), the machine needs only the first \( m + 1 \) inputs \( I_0, \ldots, I_m \) (for some \( m \)). Assume \( x' \in I_m \). Then for some \( I'_k \) \( (k > m) \) the sequence \( I_0, I_1, \ldots, I_m, I'_{m+1}, \ldots \) converges to \( x' \). But also for this input \( M \) will yield an output beginning with \( J_0, J_1, \ldots, J_n \). This means \( S(x') \in J_n \). Therefore, \( S(x') \in J_n \) for all \( x \in I_m \); hence \( S \) is continuous.

This result is not too surprising, since we have built the topology \( \tau_{\mathbb{R}} \) of the real line into the definition of \( \sigma: \sigma \) is a subbase (even a base) of \( \tau_{\mathbb{R}} \). If, for example, we replace our set \( \sigma \) by the set \( \sigma' \) of closed intervals with rational boundaries, we obtain another computability theory on the set \( \mathbb{R} \) of real numbers. Again computable functions are continuous, but now w.r.t. the finer topology, which has the closed intervals with rational boundaries as basic open sets. However, this computability theory and many similar ones have no applications.

3. COMPUTATIONAL COMPLEXITY OF REAL FUNCTIONS

In this section we outline only some basic ideas. For more details see [Ko91, Wei95b, Wei97]. By \( A^\ast \) we denote the set of all finite and by \( A^\omega \) the set of all infinite sequences with elements from \( A \).
Let \( M \) be a machine which computes a real function \( f: \subseteq \mathbb{R} \to \mathbb{R} \) as described in Section 2. For any input sequence \( I_0, I_1, \ldots \) of intervals converging to some \( x \in \mathbb{R} \) and any precision \( n \) let the computation time be the number of computation steps, which the machine \( M \) needs, until some interval \( J_k \in \sigma \) with \( \text{length}(J_n) < 2^{-n} \) has been produced. We wish to define a time complexity \( \text{Time}(x, n) \), which depends only on the real number \( x \) but not explicitly on the input sequence \( I_0, I_1, \ldots \) of atomic properties converging to \( x \). Unfortunately, the maximum of the time over all sequences \( I_0, I_1, \ldots \) converging to \( x \) is infinite in general, since rational numbers with arbitrary long notations are admitted for each of the intervals \( I_n \). On the other hand, fixing a single normalized name \( I_0, I_1, \ldots \) for each real number leads to a different unnatural computability concept for real functions (Theorem 4.6 in [Wei95b]).

The solution is a weak normalization: for any real number \( x \) restrict the set of sequences \( I_0, I_1, \ldots \) which can be used as “names” for \( x \) to a compact subset, i.e., a finitely branched tree of infinite sequences. A simple formal solution is given by the representation \( \rho: \subseteq \Sigma^\omega \to \mathbb{R} \) of the real numbers by infinite binary fractions with the (redundant system of) digits 0, 1, and \(-1\), where \( \text{dom}(\rho) = (\{0, 1, -1\}^\ast, \{0, 1, -1\}^\omega) \setminus \{0, -11, 1 - 1\}^\omega \) and

\[
\rho(b_m \cdots b_0. b_{-1} b_{-2} \cdots) = \sum_{i \leq m} b_i \cdot 2^i.
\]

If \( \rho(p) = x \), we call \( p \) a \( \rho \)-name of \( x \). With any element \( p = b_m \cdots b_0. b_{-1} b_{-2} \cdots \in \text{dom}(\rho) \) we can associate a sequence \( K_0, K_1, \ldots \) of nested intervals as follows:

\[
K_0 = [z - 1; z + 1] \text{ where } z = \sum_{i=0}^{m} b_i \cdot 2^i;
\]

for \( n > 0 \), \( K_n \) is the left (middle, right) half of \( K_{n-1} \) if \( b_{-n} = -1 \) (\( b_{-n} = 0 \), \( b_{-n} = 1 \)). Obviously, \( \{x\} = \bigcap K_n \). This somewhat unusual number system has already been used by Avizienis [Avi61], Wiedmer [Wie80] and others. We mention, merely, that ordinary infinite binary fractions (with digits 0 and 1) induce an unnatural computability concept; not even addition becomes computable under this representation [Wei95b].

For defining cost or computational complexity we need a precise model of computation. We consider generalized Turing machines which have one-way infinite input and output tapes. Notice that at least for finite inputs and outputs, Turing machine time is generally accepted as a realistic model of computational complexity.

**Definition 1.** Consider \( f: \subseteq \mathbb{R} \to \mathbb{R}, X \subseteq \text{dom}(f), s: \omega \to \omega, \) and \( t: \omega \to \omega; \) let \( M \) be a Turing machine with an infinite input tape and an infinite output tape. The machine \( M \) computes the function \( f \) on the set \( X \) in time \( t \) with
input lookahead $s$, iff for every input $p \in \rho^{-1}X$ it produces some output $q \in \Sigma^\omega$ such that

1. $f(x) = \rho(q)$,
2. $M$ writes the $n$th symbol of $q$ after at most $t(n)$ steps,
3. when $M$ writes the $n$th symbol of the output $q$, at most the first $s(n)$ symbols from the input $p$ have been read.

An easy consideration shows that the real function $f$ restricted to $X$ is Grzegorczyk computable, iff some Turing machine $M$ computes $f$ on $X$. The input lookahead measures the amount of information which is used for determining the result with given precision. It is a modulus of continuity. For a numerical value of the information in bits, we must take into account a small redundancy of the representation $\rho : \leq \Sigma^\omega \rightarrow \mathbb{R}$. Notice that the first $n$ output symbols determine the result $f(x)$ with error $c \cdot 2^{-n}$ (for some $c \leq \log |f(x)|$). The definition can be generalized easily to functions $f : \leq \mathbb{R}^n \rightarrow \mathbb{R}$. One obtains, for example: on any compact subset $X \subseteq \mathbb{R}^2$, addition can be computed in time $O(n)$ with input lookahead $n+1$ and multiplication can be computed with input lookahead $2n$ in time $O(n \cdot \log n \cdot \log \log n)$ [Wei95b] and with input lookahead $n+c$ in time $O(n \cdot \log^2 n \cdot \log \log n)$ [Sch96]. As in discrete complexity theory, the computational complexity of algorithms depends crucially on the way the input and output data are presented. Some computationally sound representations of the real numbers induce no reasonable complexity theory (see above). The computational complexity on the real numbers induced by the modified binary representation $\rho$ is sometimes called “bit-complexity.” It has been considered by many authors (Brent, Ko, Schönhage, Müller, Weihrauch) and seems to be the most important one.

4. COMPUTABILITY ON $C[0; 1]$

An IBC algorithm for a problem $S : C[0; 1] \rightarrow G$ requires pieces of partial information about the input $f \in C[0; 1]$ supplied by an oracle. Often, these pieces are function values $f(t)$ for given $t \in [0; 1]$, i.e., pairs $(t, y)$ with $y = f(t)$ which, however, are infinite objects. As in the case of real numbers, for the new model we need a countable supply $\sigma$ of “finite” pieces of information or properties about functions $f \in C[0; 1]$. A very natural choice are pairs of rational intervals $(I, J)$ with $f(I) \subseteq J$ instead of points $(t, y)$ with $f(t) = y$. We define more precisely

$$\sigma_C := \{A(a, b, c, d) \mid a, b, c, d \in \mathbb{Q}, a \leq b, c < d\},$$

where

$$A(a, b, c, d) := \{f \in C[0; 1] \mid f[a; b] \subseteq (c; d)\}.$$
Therefore, a function $f \in C[0; 1]$ has the atomic property $A(a, b, c, d)$, iff $f[a; b] \subseteq (c; d)$. An easy consideration shows that each $f \in C[0; 1]$ can be identified by its atomic properties:

$$\{ A \in \sigma_C \mid f \in A \} = \{ A \in \sigma_C \mid g \in A \} \Rightarrow f = g.$$ 

The choice of $f[a; b] \subseteq (c; d)$ instead of $f(a; b) \subseteq (c; d)$, $f(a; b) \subseteq [c; d]$ or $f[a; b] \subseteq [c; d]$ can be justified by the topology generated on $C[0; 1]$ by $\sigma_C$ as a subbase. It is the well-known compact-open topology [Eng89], which can also be generated by the maximum metric $d(f, g) := \max \{ |f(t) - g(t)| \mid t \in [0; 1]\}$.

As a further remarkable property, $\tau_C$ is the smallest topology $\tau$ on $C[0; 1]$ for which the apply function $\text{App} : C[0; 1] \times \mathbb{R} \to \mathbb{R}$, $\text{App}(f, x) := f(x)$, is continuous.

In TTE, by defining a set $\sigma$ of atomic properties we fix formally speaking a topology $\tau_\sigma$ and informally speaking a meaning of “approximation” on the set under consideration. The meaning of “effective handling of atomic properties” is fixed by a notation $\nu : \subseteq \Sigma^* \to \sigma$ of the set $\sigma$ of atomic properties. For simplicity, we do not define here a notation of $\sigma_C$ explicitly but assume that our machines can read (and write) quadruples of rational numbers and can perform the usual operations on rational numbers.

As examples we study computability of two simple IBC-problems on $C[0; 1]$: integration and determination of zeroes. On the real numbers we consider the type of computability introduced in Section 2. Then a function $S : F \to \mathbb{R}$, where $F \subseteq C[0; 1]$ is computable, iff there is some machine $M$ with the following property. If $M$ obtains a list $I_0, I_1, \ldots$ of all atomic properties $I \in \sigma_C$ of some $f \in F$ then it produces a list $J_0, J_1, \ldots$ of open intervals converging to $S(f)$. As in the case of real functions, an easy consideration shows that such a computable function $S$ must be $(\tau_C, \tau_\mathbb{R})$-continuous, which is a special case of the general Theorem 4.4 in [Wei95b].

We describe a machine, computing the integral of a function $f \in C[0; 1]$. By assumption, an input for $M$ is a sequence $(a_i, b_i, c_i, d_i)_{i \in \omega}$ of all rational quadruples $(a, b, c, d)$ with $f[a; b] \subseteq (c; d)$ (for some $f \in C[0; 1]$). Consider $n \in \omega$. By continuity of $f$, for any $x \in [0; 1]$ there are rational numbers $a_x, b_x, c_x, d_x, e_x, g_x$ such that $a_x < e_x < x < g_x < b_x, c_x < d_x < c_x + 2^{-n}$ and $f[a_x; b_x] \subseteq (c_x; d_x)$. Since $[0; 1]$ is compact, $[0; 1] \subseteq \bigcup V$ for some finite subset $V \subseteq \{(e_x; g_x) \mid x \in [0; 1]\}$. Therefore, there is a finite subset $B \subseteq \omega$ such that $[0; 1] \subseteq \bigcup \{(a_i; b_i) \mid i \in B\}$ and $0 < d_i - c_i < 2^{-n}$ for all $i \in B$. Let the machine $M$ operate in stages $n, n \in \omega$ as follows: In Stage $n, M$ searches a finite set $B \subseteq \omega$ such that $[0; 1] \subseteq \bigcup \{(a_i; b_i) \mid i \in B\}$ and $d_i - c_i < 2^{-n}$ for all $i \in B$. From these data, $M$ determines rational numbers $r_n$ and $s_n$ with $r_n < \int f < s_n < r_n + 2^{-n}$ and gives the interval $(r_n; s_n)$ to the output. Obviously, $M$ produces an infinite sequence $J_0, J_1, \ldots$ of intervals converging to $\int f$.

By the classical intermediate value theorem any function from $H := \{ f \in C[0; 1] \mid f(0) = -1 \text{ and } f(1) = 1 \}$ has a zero. The theorem can be proved
by the bisection method which immediately gives a fast IBC-algorithm for determining zeroes. Notice that such an algorithm uses branchings of the type “\( f(x_m) > 0 \)” with real \( x_m \). On the one hand, these branchings are the sources of discontinuity in IBC-algorithms; on the other hand, they cannot be realized by physical machines. Furthermore, an easy proof shows [Wei95b, Her96a] that there is no \((\tau_C, \tau_{\mathbb{R}})\)-continuous function \( Z : H \to \mathbb{R} \) with \( f(Z(f)) = 0 \) for all \( f \in H \); hence such branchings must be used. By our previous remark on continuity, there is no TTE-computable function \( Z \), in particular. There is, however, a positive result for the subset

\[ H_I := \{ f \in C[0; 1] | f(0) = -1, f(1) = 1, f \text{ increasing} \} \]

of increasing functions. We apply the well-known trisection method. We sketch a TTE-algorithm which gets an input sequence \((a_i, b_i, c_i, d_i)_{i \in \omega} \) of all atomic properties \( A(a, b, c, d) \in \sigma_C \) of some function \( f \in H_I \) and produces a sequence of Intervals \( J_0, J_1 \ldots \) converging to the unique zero of \( f \).

Let \( I_0 := [0; 1] \). Assume that \( J_n = [l_n; r_n] \) has been determined, such that \( f(l_n) < 0 \) and \( f(r_n) > 0 \).

Determine \( m_l := l_n + (r_n - l_n)/3 \); \( m_r := r_n - (r_n - l_n)/3 \). (Then \( m_l \neq 0 \) or \( m_r \neq 0 \).) Search for some \( k \) such that \( m_l \in [a_k; b_k] \) or \( m_r \in [a_k; b_k] \) and \( 0 \notin (c_k; d_k) \) (this search ends successfully, since at least one of the values \( f(m_l) \) and \( f(m_r) \) is different from 0):

\[ J_{n+1} := \begin{cases} [m_l; r_n], & \text{if } d_k < 0 \\ [l_n; m_r], & \text{otherwise.} \end{cases} \]

(Then, \( f(l_{n+1}) < 0 \) and \( f(r_{n+1}) > 0 \) and \( r_{n+1} - l_{n+1} = 2(r_n - l_n)/3 \).)

Obviously, a machine working according to this method determines the zeroes for functions from \( H_I \). A generalization of this method is applicable to the subset \( H_{ND} \) of functions from \( H \) for which the set of zeroes is nowhere dense. However, in this case the algorithm cannot be “extensional”; i.e., different inputs converging to the same function may be mapped to different zeroes. For a detailed discussion of the zero-finding problem on \( C[0; 1] \) see [Wei87, Wei95b, Her96a]. The separation of discontinuous problems into continuous subproblems has been studied by Hertling [Her96b].

In the above discussion we considered atomic properties of type \( f[a; b] \subseteq (c; d) \). Often properties about derivatives of \( f \) are also available. For \( f \in C^k[0; 1] \) one may consider all properties \( f^{(m)}[a; b] \subseteq (c; d) \) \((m \leq k, a, b, c, d \in \mathbb{Q})\) as atomic. Notice that for \( k > 0 \) these atomic properties generate a topology \( \tau_{Ck} \) which is finer than \( \tau_C \) on \( C^k[0; 1] \).
As in the case of real functions, lists of all atomic properties \( A \in \sigma_C \) which hold for \( f \in C[0; 1] \) are not adequate names for studying computational complexity of functions \( S: C[0; 1] \to G \). The solution proposed for real functions in Section 3 can be generalized from \( \mathbb{R} \) to compact (or even locally compact) sets. Informally speaking, the set of names of the points for each compact space can be restricted to a set \( D \) of infinite sequences of atomic properties forming a finitely branched infinite tree. The idea of the construction is simple. Let \( F \) be a compact metric space. Then for each \( n \in \omega \) there is a finite set \( \beta_n \) of open balls with radius \( 2^{-n} \) which covers \( F \). Let \( D \) consist of all sequences \( I_0, I_1, \ldots \) such that \( I_n \in \beta_n \) for all \( n \in \omega \) and \( \bigcap_{n \in \omega} I_n \neq \emptyset \). Indeed, these sequences (extended by \( I_{-1} := F \)) form a finitely branched tree. The approximation concept on the set \( F \) induced by this naming of the elements by sequences from \( D \) is consistent with the metric on \( F \); i.e., the representation is “admissible” [Wei87]. Obviously, the sets \( \beta_n \) of pieces of atomic information and the notation \( \nu \) of their union fix the computability and complexity theory on the compact space \( F \).

As an example for a formal treatment we consider the set

\[
F := \{ f \in C[0; 1] \mid f(0) = 0 \text{ and } |f(x) - f(y)| \leq |x - y| \text{ for } x, y \in [0; 1] \}
\]

of Lipschitz bounded functions. Indeed, \( F \) is a compact subset of the metric space \((C[0; 1], d)\). We define explicitly a representation \( \delta_L : \Sigma^\omega \to F \) as follows. For any \( w = a_1a_2 \ldots a_k \in \{0, 1, -1\}^* \) let \( P(w) \in C[0; 1] \) be the polygon function determined by the break points \((i/k, y_i), (0 \leq i \leq k)\) with \( y_0 = 0 \) and \( y_i = y_{i-1} + a_i/k \). For \( f \in F \) and \( q \in \Sigma^\omega \) define

\[
\delta_L(q) = f : \iff \exists w_0, w_1, \ldots \in \Sigma^* \text{ such that } q = w_0w_1\ldots \text{ and } w_n \in \{0, 1, -1\}^{2^n} \text{ and } d(P(w_n), f) \leq 2^{-n} \text{ for all } n.
\]

Notice the similarity of \( \delta_L \) with the representation \( \rho \) of the real numbers from Section 3. We state here without proof that Turing machine computations on \( \delta_L \)-names induce the computability theory on \( F \), which we have introduced informally in Section 4.

Integration of functions from \( F \) using \( \delta_L \)-names is an easy task for a Turing machine. Since \( d(f, P(w_n)) \leq 2^{-n} \), we have \( |\int f - \int P(w_n)| \leq 2^{-n} \). For determining the integral with precision \( 2^{-n} \), it suffices to read \( w_n = a_0 \cdots a_{2^n} \) and determine \( \int P(w_n) \) which can be done in time \( O(n \cdot 2^n) \), the time needed for adding or subtracting \( 2^n \) binary numbers of length \( n \). As a result we obtain:

**Theorem 2.** Integration on \( F \) can be computed w.r.t. \((\delta_L, \rho)\) in time \( O(n \cdot 2^n) \) with input lookahead \( c \cdot 2^n \), where \( c \) is independent of the particular Turing machine model.
Notice, that also for the representation \( \delta_L \) the input lookahead is, except for some small redundancy factor, the number of bits of information which is needed to determine the result with precision \( 2^{-n} \).

The next problem we consider is zero-finding. For a detailed discussion of the complexity of zero-finding on \([0; 1]\) in the IBC-model of computation see [NW96, Her96b]. From Section 4 we know, that on the set \( H_I \) of increasing functions \( f \in C[0; 1] \) with \( f(0) = -1 \) and \( f(1) = 1 \) zeroes can be determined by a machine, which gets atomic properties from \( \sigma_C \) as input and produces a sequence of nested intervals as output. For reasons which we have already discussed for the case of real numbers, algorithms with such inputs cannot be used for studying computational complexity.

Furthermore, the set \( H_I \) contains functions which are extremely flat in the neighborhoods of their zeroes. For such a function \( f \), for computing the zero with precision \( \varepsilon \) approximate values \( f(x_m) \) of much higher precision are necessary, the determination of which requires much input information and consequently much computation time for reading and handling them. A bound uniform over \( H_I \) does not exist.

According to our general strategy we have to consider compact domains and to use adequate representations. The set \( H_I \) is not compact since it is not even closed in \((C[0; 1], d)\). For example, the piecewise linear function \( f_0 \) with the vertices \((0, -1), (1/3, 0), (2/3, 0), (1, 1)\), which is zero on \([1/3; 2/3]\), hence not increasing, is not in \( H_I \) but in its metric closure.

For avoiding further technical definitions, in the next example we consider the determination of points with value 1/2 instead of 0. Let \( F \) be the set of Lipschitz bounded functions with the representation \( \delta_L \) defined above. Then for every increasing function \( f \in F \) with \( f(1) \geq 1/2 \) there is some number \( x \) with \( f(x) = 1/2 \). The subset of these functions, however, is not compact. We obtain a compact subset of \( F \) by bounding the modulus of continuity also of \( f^{-1} \), where a function \( m: \omega \rightarrow \omega \) is called a modulus of continuity of \( f \in C[0; 1] \), iff \(|x - y| \leq 2^{-m(n)} \implies |f(x) - f(y)| \leq 2^{-n} \). For any function \( m: \omega \rightarrow \omega \) we define

\[
F_m := \{ f \in F \mid f \text{ increasing, } f(1) \geq 1/2, \text{ and } |f(y) - f(x)| \leq 2^{-m(n)} \implies |y - x| \leq 2^{-n} \}.
\]

As a closed subset of the compact set \( F \), \( F_m \) is also compact.

Consider \( q = w_0w_1w_2 \cdots \in \Sigma^\omega \), \( f \in F_m \), and \( n \in \omega \) with \( \delta_L(q) = f \). We wish to determine a \( 2^{-n} \)-approximation of the unique \( x_0 \in [0; 1] \) with \( f(x_0) = 1/2 \). We choose \( k := m(n + 1) + 2 \) and define \( q_k := P(w_k) \). If \(|x - x_0| < 2^{-k}\), then \(|q_k(x) - 1/2| \leq |q_k(x) - f(x) + f(x) - 1/2| \leq 2 \cdot 2^{-k}\). Therefore, there is some \( x_1 = N \cdot 2^{-k} \), \( N \in \{0, 1, \ldots, 2^k\} \), such that \(|q_k(x_1) - 1/2| \leq 2 \cdot 2^{-k}\). We obtain \(|f(x_1) - f(x_0)| \leq |f(x_1) - q_k(x_1) + q_k(x_1) - 1/2| \leq 3 \cdot 2^{-k} < 2^{-(k-2)} = 2^{-m(n+1)}\). The assumption \( f \in F_m \) implies \(|x_1 - x_0| < 2^{-(n+1)}\).
Assume now that \( m: \omega \to \omega \) is computable in time \( O(2^{m(n+1)}) \) on a Turing machine. Then there is some Turing machine, which for input \( q = w_0 w_1 w_2 \ldots \in \text{dom}(\delta_L) \) for each \( n = 0, 1, 2, \ldots \) determines \( k := m(n+1) + 2 \) and some \( N_n \in \{0, \ldots, 2^k\} \) such that \( |P(w_k)(N_n \cdot 2^{-k}) - 1/2| < 2^{-k} \). As we have shown above, \( N_n \) exists and \( N_n \cdot 2^{-k} \) approximates the 1/2-argument of \( f = \delta_L(q) \) with error of less than \( 2^{-n-1} \). With some easy additional considerations we obtain the following theorem.

**Theorem 3.** Let \( m: \omega \to \omega \) be an increasing function such that \( m(n) \) can be computed on a Turing machine in time \( O(2^{m(n+1)}) \). Then for any \( f \in F_n \) the zero of \( f(x) - 1/2 \) can be computed w.r.t. the representations \((\delta_L, \rho)\) in time \( O(2^{m(n+1)}) \) with input lookahead \( c \cdot 2^{m(n+1)} \), where \( c \) does not depend on the particular Turing machine model.

In both examples, integration and zero-finding, the time complexity is not much greater than or of the same order as the input lookahead. Roughly speaking, the time is consumed already for reading the input information, while the computations are easy. The amount of information bounded by the input lookahead is not only sufficient, but also necessary, for determining the result with precision \( 2^{-n} \). In both examples, the computation time of an algorithm may depend crucially on the way the input and output data are presented. The representation \( \delta_L : \subseteq \Sigma^\omega \to F \) of the function space used here seems to be quite natural and may appear in many applications. But also other cases might be of interest, e.g., representations by converging sequences of rational polynomials etc.

Consider integration and zero-finding in IBC. We assume that the information about the input \( f \in F \) is of the form \( N(f) = (f(t_1), \ldots, f(t_j)) \). A \( 2^{-n} \)-approximation of \( \int f \) can be obtained by the quadrature formula \( Q_{2^n} := 2^{-n} \sum_{i=1}^{2^n} f((2i-1)/2)/2 \cdot 2^n \). The complexity of an IBC-algorithm based on this formula is in \( O(2^n) \). Notice, that for determining a \( 2^{-n} \)-approximation of the integral the exact values \( f(t_i) \) are not necessary but only the first \( n \) digits of the representation by infinite binary fractions of each of them. Therefore, \( Q_{2^n} \) can be computed in \( O(n \cdot 2^n) \) bit operations on a Turing machine. We obtain the same bit-complexity in Theorem 2. The bisection method for zero-finding on the set \( H := \{f \in C[0; 1] \mid f(0) = -1 \text{ and } f(1) = 1\} \) can be formulated as an IBC-algorithm in which \( n \) function evaluations are sufficient for a \( 2^{-n} \)-approximation of a zero, and the IBC-complexity is in \( O(n) \). Since tests \( f(t) \leq 0 \) are used, the IBC-algorithm, however, cannot be realized on a computer. In fact, any partial function \( Z : \subseteq C[0; 1] \to \mathbb{R} \) determining a zero of each \( f \in H \) is not continuous and, hence, cannot be TTE-computable. However, the restriction of \( Z \) to the set \( H_I \) of increasing functions, which change their sign, is continuous and even computable [Wei95b]. In TTE, for obtaining a uniform complexity bound we need a further restriction of \( Z \) to a compact subset \( H' \) of \( H_I \). Such a set \( H' \) has a uniform modulus of continuity, as well as the set of
its inverses. Both conditions are used in Theorem 3. The trisection algorithm as an IBC-algorithm requires at most $2n$ function evaluations $f(t_i)$ and has a complexity in $O(n)$. For each of the numbers $f(t_i)$ only finitely many bits of their infinite binary representations are needed. However, the number of these bits may increase rapidly with $n$ if the function is very “flat,” i.e., if the inverse of $f$ has no small modulus of continuity. A detailed analysis gives a bit complexity which is similar to that in Theorem 3. In this situation, the unit cost assumption for IBC-algorithms is no longer realistic.

6. CONCLUSION

The discussion in Section 1 and the example of zero-finding explain why the model of computation currently used in information based complexity is unrealistic in many cases. Any statement proved in IBC concerning the behavior of an algorithm remains useless for practice, unless some additional argument or investigation shows that the algorithm can be realized sufficiently reliably by a computer. However, without a mathematical formulation expressing the ability of real computers such an investigation must remain informal. In this paper we propose a refined model of computation for IBC which is very realistic and therefore can serve as such a mathematical formulation.

The model can be obtained from the IBC-model of computation by the restriction that machines can handle, i.e., read, store, write, etc., only finite properties of in general infinite objects. Turing machines, which are already used in discrete computability and complexity theory as a very realistic model, are considered as the mathematical computing devices. The resulting computability and complexity theory is the well-known “type 2 theory of effectivity.” In TTE, computability on a set $M$ is introduced via a representation which in most applications can be derived from an information structure $(M, \sigma, \nu)$, where $\sigma$ is a countable set of subsets of $M$ (the “atomic properties”) identifying points of $M$, and $\nu$ is a notation of $\sigma$. The set $\sigma$ specifies a concept of “approximation” on $M$ and $\nu$ specifies computability. If $M$ is, e.g., the input set for an algorithm, the elements of $\sigma$ are those pieces of finite information, which are available as inputs for approximating elements of $M$, and the notation $\nu$ specifies, how these pieces are encoded by finite words.

For a set $M$ like $\mathbb{R}$ or $C[0; 1]$ there are only very few computability theories relevant for applications. We have considered the most important ones in this paper (with the real line topology on $\mathbb{R}$ and the compact-open topology on $C[0; 1]$). Various modifications of $\sigma$ and $\nu$ leave these computability theories unchanged. On the other hand, computational complexity is very sensitive to minor changes of the machine model and the way input and output data for machines are presented, in particular for small complexity bounds, e.g. $n \cdot \log n$. Therefore, in TTE there may be various nonequivalent complexity definitions.
which are relevant for practical applications. In our examples we have specified complexity theories on $\mathbb{R}$ and $C[0; 1]$ by choosing the representations $\rho : \subseteq \Sigma^\omega \rightarrow \mathbb{R}$ and $\delta_L : \subseteq \Sigma^\omega \rightarrow C[0; 1]$, respectively. This type of computational complexity for real functions, sometimes called “bit-complexity,” has been considered by many authors [Bre76, Alt85, Ko91, Mül86, Mü87, Sch90, Wei91, Sch96, NR96].

In TTE, every computable function is continuous. In IBC, however, computability and topology on a set are (almost) unrelated. In fact, most functions computed by IBC-algorithms are not continuous. Discontinuity is caused by tests $x < 0$ etc., which are basic operations in IBC. Many problems of practical interest are discontinuous and can be solved by IBC-algorithms or real RAMs, e.g., zero-finding on $C[0; 1]$ or most of the problems in computational geometry [PS85]. Consider, e.g., the real function $s$ with $s(x) := (0$ if $x < 0; 1$ otherwise). Obviously, the function $s$ is IBC-computable but not continuous and not TTE-computable. While every TTE-algorithm can be realized correctly on a digital computer, no physical device is known until today which realizes arbitrary IBC-algorithms and, in particular, the function $s$.

For applications, however, realizable algorithms are needed which approximate such discontinuous functions in some reasonable way. First of all the discontinuities of a given problem must be localized. A level of discontinuity can be defined in a straightforward abstract way which is closely related to the number of tests needed in “continuous real RAMs” ([HW94] and the comprehensive study [Her96b]). Our real function $s$ has 0 as its only point of discontinuity. We discuss some TTE-computable and hence realizable substitutes for $s$.

Let $s_1$ be the restriction of $s$ to the set $\mathbb{R}\setminus\{0\}$. There is some TTE-algorithm, which computes $s_1(x)$ for all inputs $x \neq 0$ and diverges for input 0. For obtaining a uniform complexity bound, the function $s$ must be restricted to some compact set $[a; b] \cup [c; d]$ with $b < 0 < c$. Such restrictions, however, are not acceptable in applications for which 0 is a possible input parameter. Another substitute for the function $s$ is a continuous and computable function “close” to $s$, e.g. $s_2(x) := \tanh(n \cdot x)$ for some large number $n$ depending on the concrete application. But this function is no longer 0-1-valued and therefore useless in many applications. The last substitute for $s$ we propose is a function $s_3$ which becomes many-valued or “nondetermined” in some small neighborhood $[-\delta; \delta]$ of 0 (where $\delta > 0$ has to be chosen by the user), i.e., $s_3(x) = 0$ if $x < -\delta$, $s_3(x) \in \{0, 1\}$ if $|x| \leq \delta$ and $s_3(x) = 1$ if $x > \delta$. There is indeed a TTE-algorithm which computes a function $s_3$ of this type and which is realizable on a digital computer. For an input $x$ with $|x| \leq \delta$ the result may be 0 or 1, depending on the kind of input information supplied to the algorithm or computer. In mathematical terms, the algorithm operating on “names” of real numbers is not extensional on the interval $[-\delta; \delta]$.

In many important applications such locally many-valued functions can be used as substitutes for discontinuous functions. Consider the computation of an approximate zero of a function by iteration. Repeat the iteration step if the error
is greater than $2\delta$, stop if the error is less than $\delta$, and repeat or stop otherwise. Consider a car moving toward a wall. Go on if the distance is greater than 2 cm, stop if it is less than 1 cm, and go on or stop otherwise. As a last example consider zero-finding of a function $f_c$, $f_c(x) := x^3 - x + c$, as a function of $c$. This problem has no continuous or computable function as its solution, but it can be solved by a TTE-algorithm, which is two-valued in some neighborhood $[-\delta; \delta]$ [Wei95b, Bra96, Wei97].

In this paper we have presented another model of computation, the TTE-model, for numerical analysis. The real-number model is very simple but unrealistic in many situations. The TTE-model, on the other hand, is very realistic, but in general programming is more sophisticated, since problems of discontinuity, instability etc., must be considered by the programmer in advance. Although Turing machines are used for the mathematical formulation of the model, in practice higher level programming languages can be used instead of the somewhat unwieldy Turing machines. Further investigations should give conditions under which IBC-algorithms are realistic, where TTE-algorithms can be used for a mathematical definition of “realistic.” Recently, Brattka and Hertling [BH96] have proposed a modified real RAM which admits easy programming, which is realizable on digital computers and additionally has a realistic computational complexity.

ACKNOWLEDGMENTS

I thank Vasco Brattka, Peter Hertling, and Erich Novak for many discussions and helpful comments.

REFERENCES


