数理解析研究所講究録
1169 巻 2000 年 62-75

Realistic models of computability on the real numbers

Vasco Brattka
Theoretische Informatik I, FernUniversität Hagen, D-58084 Hagen, Germany
e-mail: vasco.brattka@fernuni-hagen.de

April 26, 2000

Abstract

In this paper we discuss the role of the model of computability in the cycle of real number programming. We argue that problems occurring in practice, like instability or degeneracy problems, are caused by the real number model. As a more realistic approach we propose the point of view of computable analysis and we present a feasible real number model which is compatible to this approach. Finally, we briefly discuss a general theory of continuous data structures which is based on computable analysis.

Keywords: real number model, computable analysis, continuous data structures.

1 Introduction

We will start to discuss the role of the model of real number computation in the cycle of real number programming. Typically, this cycle can be characterized by the following three steps (cf. Figure 1):

(1) **Description:** in a first step the problem under consideration is described in terms which are offered by some given model of computation on the real numbers.

(2) **Programming:** in a second step the model of computation is used to develop some algorithm which solves the problem. Thus, "programming" here means "developing algorithms".

(3) **Implementation:** last not least, the algorithm is implemented in some concrete programming language on a physical computer.

It is easy to realize that this cycle of real number programming sensitively relies on the chosen model of computability on the real numbers. This model already determines the description of the problem, as well as the construction of the algorithm. Moreover, it depends on the model of computability whether a correct implementation of the algorithm on physical computers will be possible or not.

It is the *real number model* which is used as the standard model of computability on the real numbers in various parts of computer science and mathematics. This model is

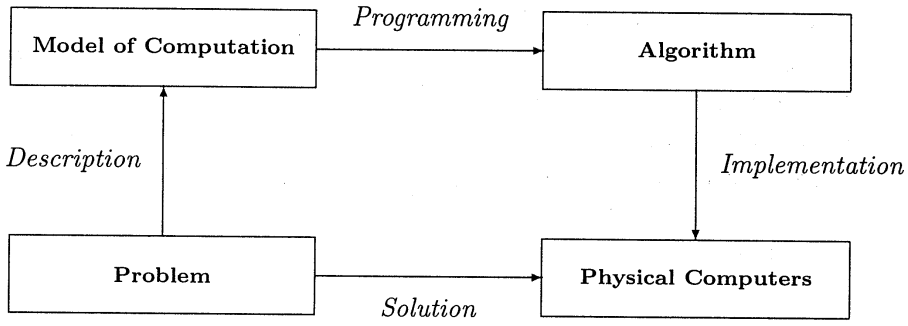

Figure 1: The cycle of real number programming

based on the idea that real numbers are entities and formally it is introduced via register machines or so-called real random access machines which are equipped with real number registers and which can perform arithmetic operations as well as comparisons and equality tests on real numbers precisely.

The real number model is the standard model in computational geometry (cf. Preparata and Shamos [PS85]); it has been used by Blum, Shub and Smale [BSS89] to introduce a theory of computational complexity and by Traub, Wasilkowski and Woźniakowsky [TWW88] to describe the complexity of algorithms in numerical analysis.

From a certain point of view the real number model is the mathematical formalization of the semantics which is offered by typical imperative programming languages on the real numbers. Languages like JAVA, C, PASCAL and FORTRAN offer a data type "real" together with arithmetic operations and comparisons. But unfortunately, this semantics cannot be realized precisely on physical computers. This is not just a precision problem of floating point arithmetic but it is a general limitation of physical computers.

Heuristically, the problems of the real number model are well-known to practitioners, e.g. numerical analysts know that comparisons with zero can be problematic and have to be avoided. Special terminologies have been invented to describe the problems: the problem under consideration can be called *ill-conditioned*, or a specific input can be called *degenerated*, or the algorithm can be called *instable*.

From the point of view of computable analysis the cardinal problem is that real numbers are infinite objects and in finite time we can only handle finite portions of information. Consequently, *discontinuous* problems cannot be solved by physical machines. Especially, the precise comparisons and equality tests of the real number model cannot be performed by physical computers.

The importance of the discontinuity problem depends on the specific area of application. For instance, in numerical analysis most of the algorithms treat continuous problems like numerical integration; discontinuous problems like numerical differentiation have been realized as unsolvable in the general case. Even if an algorithm uses some discontinuous test (like the Heron algorithm) this causes no problems in practice, since the corresponding problem is continuous (like the square root function). The situation in computational geometry is quite different. There the problems under consideration are typically discon-

tinuous (e.g. test whether a point belongs to some convex hull) and thus they cannot be solved without the discontinuous basic operations of the real number model (cf. Hertling and Weihrauch [HW94] and Burnikel, Mehlhorn, and Schirra [BMS94]). In the worst case this can lead to algorithms which map typical inputs to completely incorrect outputs. This phenomenon can also be illustrated by the function

$$f :\subseteq \mathbb{R} \to \mathbb{R}, (x,y) \mapsto \begin{cases} 1 & \text{if } x \in \mathbb{Q} \\ 0 & \text{if } x \notin \mathbb{Q}, x^2 \in \mathbb{Q} \end{cases},$$

undefined in all other cases, which can be computed according to the real number model but which can neither be approximately computed in a realistic sense nor be implemented correctly using floating point arithmetic (cf. Weihrauch [Wei98]).

This highly unsatisfactory situation could be resolved by a Copernican turn: instead of developing the best possible algorithms according to the real number model, one should search for the best possible model according to the abilities and limitations of physical computers. As soon as such a model has been discovered, computer scientists should feel free to replace the semantics of imperative programming languages by a semantics which perfectly fits together with the new model.

Computable analysis is the right tool to figure out such a realistic model because it is based on Turing machines. Following Church's thesis, we can assume that Turing machines describe the possibilities of handling finite information by physical machines in the most realistic way according to our current knowledge.

We close the introduction with a short survey on the organization of the paper: in the following Section 2 we will shortly discuss the model of computability which is used in computable analysis. In Section 3 we present the feasible real random access machine model, which has been introduced in a joint project with Peter Hertling [BH98] and which is a realistic modification of the real number model with respect to computability and complexity. Finally, in Section 4 we briefly discuss a general theory of continuous data structures and we sketch how the ideas that have been applied to the real numbers can be transferred to other objects like compact sets and continuous functions.

2 Computable analysis

In this section we would like to present some basic definitions and results from computable analysis which show how a realistic Turing machine based model of computability on the real numbers can be defined. The origins of computable analysis go back to Turing's paper [Tur36] in which he presented his famous machine model. It is not very well-known that one of his purposes was to introduce a formal definition for the notion of a computable real number. The corresponding notion of a computable real number function has systematically been studied by Grzegorczyk [Grz57] and Lacombe [Lac55] in the fiftieth. Later on, the theory of computable analysis has been developed by Hauck [Hau73], Pour-El and Richards [PER89], Ko [Ko91] and Kreitz and Weihrauch [KW84] and many others. The presentation here will be mainly based on Weihrauch [Wei00]. The basic idea

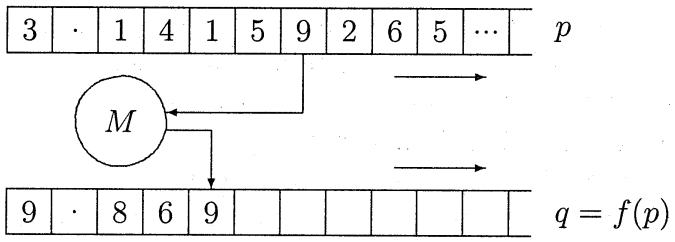

Figure 2: A Turing machine computing the square function $x \mapsto x^2$

of computable analysis can be described in the following way: fix some representation of the real numbers by infinite sequences and call a real number function computable, if there exists a Turing machine that transfers each sequence which represents some input into a sequence which represents the corresponding output of the function. Figure 2 illustrates the situation for the square function with input π in decimal representation. More formally, we will first fix the notion of a representation.

Definition 2.1 (Representation) A *representation* of the real numbers $\mathbb{R}$ is a surjective function $\delta :\subseteq \Sigma^\omega \to \mathbb{R}$.

Here Σ^ω denotes the set of infinite sequences over the alphabet Σ the inclusion symbol "$\subseteq$" indicates a potentially partial functions. An example of a representation is the ordinary decimal representation. Now it is straightforward to define the notion of a computable real number function.

Definition 2.2 (Computable real number function) A function $f :\subseteq \mathbb{R} \to \mathbb{R}$ is called *computable* with respect to some representation $\delta :\subseteq \Sigma^\omega \to \mathbb{R}$, if there exists some Turing machine M (with one-way output tape) which computes infinitely long and which in the long run transforms each sequence $p \in \Sigma^\omega$ which represents some $x := \delta(p) \in \mathbb{R}$ into some sequence $q \in \Sigma^\omega$ which represents $f(x)$, i.e. $f(x) = \delta(q)$.

It is straightforward how to extend this definition to multi-dimensional functions $f :\subseteq \mathbb{R}^n \to \mathbb{R}$. It is easy to observe that this definition of a computable real number function sensitively relies on the chosen representation of the real numbers. Turing, who chose the ordinary decimal representation of the real number in his first attempt, realized in a correction of his famous paper [Tur37] that the decimal representation has some serious disadvantages and other representations are preferable. Especially, he realized the following:

Proposition 2.3 (Decimal representation) *Multiplication by* 3*, i.e. the real function* $f : \mathbb{R} \to \mathbb{R}, x \mapsto 3x$ *is not computable with respect to the decimal representation.*

The proof by contradiction is quite easy: each machine which would compute multiplication by 3 w.r.t. the decimal representation has to transform the input $p = 0.33333...$

either to $q = 0.9999...$ or to $q' = 1.0000....$ Especially, the machine has to write 0.9 or 1.0 on the output tape after some finite time, but no finite prefix of the input sequence p suffices to decide whether the correct output sequence has to start with 0.9 or with 1.0. Consequently, such a machine cannot exist.

As a consequence of this phenomenon Turing did not abandon his machine model for computations with real numbers but he replaced the decimal representation by some other representation. One possible choice of some reasonable representation is the so-called "signed-digit representation" which is defined like the decimal representation but which also allows negative digits. From now on we will assume that δ is the binary signed-digit representation of the real numbers which operates with base 2 and digits $-1, 0, 1$. Computability of real number functions will be understood w.r.t. this representation. Most concrete continuous functions which are used in analysis are computable in this sense.

Proposition 2.4 (Computable real functions) *The real arithmetic operations* $+, -, \cdot, \div :\subseteq \mathbb{R}^2 \to \mathbb{R}$ *and the functions* $\sin, \cos, \exp : \mathbb{R} \to \mathbb{R}$ *are computable real number functions.*

The proof for this and the following results in this section can be found in [Wei00]. One of the main observations of computable analysis is that all computable real number functions are continuous.

Theorem 2.5 (Continuity of computable functions) *All computable real functions* $f :\subseteq \mathbb{R}^n \to \mathbb{R}$ *are continuous.*

Since each finite prefix of the output has to be *computed* from some finite prefix of the input, each finite prefix of the output has at least to *depend* on some finite prefix of the input. But this dependency is nothing but continuity. As a direct consequence one obtains that the equality test on the real numbers is not decidable.

Proposition 2.6 (Undecidability of the equality) *The equality test on the real numbers is not decidable, i.e. the function*

$$e : \mathbb{R}^2 \to \mathbb{R}, (x, y) \mapsto \begin{cases} 0 & \text{if } x = y \\ 1 & \text{else} \end{cases}$$

is not computable.

Now one could ask whether the undecidability of the equality is just a bad property of the signed-digit representation (in the same sense as the non-computability of the multiplication by 3 is just a bad property of the decimal representation). Unfortunately, there is an easy cardinality argument which shows that equality is undecidable with respect to *all* representations of the real numbers [Wei00]. In other words: there is no way to represent real numbers which could enable physical computers to decide equality!

Real number model	Computable analysis
arbitrary constants	only computable constants
arithmetic operations are computable	arithmetic operation are computable
precise tests $=, <$	only finite precision tests
sin, cos, exp are not computable	sin, cos, exp are computable

Table 1: Models of computability on the real numbers

The following table summarizes some joint properties and some differences between the real number model and the model of computability as it is used in computable analysis.

The property stated in the first line of the table is due to the fact that in the formalization of the real number model of Blum, Shub and Smale [BSS89] arbitrary constants are allowed. This enables BSS machines to decide arbitrary discrete problems (the characteristic function can simply be stored in a constant). Thus, if one insists on arbitrary constants then the model is not only unrealistic for topological reasons but also for recursion theoretic reasons. Moreover, the basic real number model is not only unrealistic but also incomplete, since many important functions (like the trigonometric functions) are not computable in this model because it does not offer any limit construction. Finally, also the class of recursive sets which have been defined with the help of the real number model is not very reasonable, since easy sets like the graph or the closed epigraph of the exponential functions are not recursive in this sense [Bra99a].

3 Feasible real random access machines

In the previous section we have seen how a realistic Turing machine based model of computability on the real numbers can be defined. One problem related to this model is that it will be difficult to convince practitioners in computational geometry or numerical analysis to describe their algorithms in terms of Turing machines. In this section we would like to introduce the so-called feasible real random access machine model (feasible RAM) which overcomes this problem since it characterizes the Turing machine approach of computable analysis in terms as close as possible to the real number model. The feasible real random access machine model as well as all presented results are due to a joint project with Peter Hertling [BH98]. The feasible real RAM can be characterized by the following features:

- rational constants,
- usual arithmetic operations on $\mathbb{N}$ and $\mathbb{R}$,
- ordinary tests $<, =$ on $\mathbb{N}$,
- finite precision test $<_k$ on $\mathbb{R}$,

- approximative semantics,
- logarithmic time complexity measure.

We will not formalize the definition of our feasible real RAMs but we will describe it intuitively. Roughly speaking, a feasible real RAM is a register machine with two types of registers: natural number registers n_i and real number registers r_j. A finite number of these registers can be used as input and output registers. The program of a feasible real RAM is controlled by some finite flowchart. A precise list of all basic operations which are allowed in flowcharts of the feasible real RAM together with there logarithmic costs is given in the following table. The logarithmic cost $\ell(x)$ denotes roughly speaking the length of the binary representation of the integer part of x. More precisely, for $x \in \mathbb{N}$ or $x \in \mathbb{R}$ we use $\ell(x) = 1 + \lfloor \log(\max\{|x|, 1\}) \rfloor$. Here log denotes the binary logarithm: $\log(x) := \log_2(x)$.

	op	cost(*op*)	explanation
assignment of constants	$n_i := m$	1	$m \in \mathbb{N}$
	$r_i := q$	1	$q \in \mathbb{Q}$
simple copy instructions	$n_i := n_j$	$\ell(n_j)$	
	$r_i := r_j$	$\ell(r_j)$	
mixed copy instructions	$r_i := n_j$	$\ell(n_j)$	
	$n_i := \lfloor r_j \rfloor_{n_k}$	$\ell(r_j, n_k)$	
natural arithm. operations	$n_i := n_j \otimes n_k$	$\ell(n_j, n_k)$	$\otimes \in \{+, \dot{-}, *, \text{div}, \text{mod}\}$
real arithmetic operations	$r_i := r_j \otimes r_k$	$\ell(r_j, r_k)$	$\otimes \in \{+, -, *\}$
	$r_i := r_j / r_k$	$\ell(r_j, \frac{1}{r_k})$	
tests	$n_i = n_j$	$\ell(n_i, n_j)$	
	$n_i < n_j$	$\ell(n_i, n_j)$	
	$r_i <_{n_k} r_j$	$\ell(r_i, r_j, n_k)$	

Table 2: Feasible RAM operations and their costs

The finite precision test $<_k$ with precision k can be defined precisely by

$$(x <_k y) \ni \begin{cases} \text{TRUE} & :\Longleftrightarrow x < y \\ \text{FALSE} & :\Longleftrightarrow x > y - \frac{1}{k+1} \end{cases}$$

for $x, y \in \mathbb{R}$, $k \in \mathbb{N}$. In other words: if the test $x <_k y$ answers with TRUE, then always $x < y$ holds; and if the answer is FALSE, then $x > y - \frac{1}{k+1}$ holds. In the small overlapping area of uncertainty of length $\frac{1}{k+1}$ the answer of the test might be TRUE as well as FALSE (cf. Figure 3). The reader should notice that the costs of the test operation increase if the length $\frac{1}{k+1}$ of the overlapping area decreases. For complexity reasons we will also use a finite precision staircase operation, giving one of the values in $\lfloor x \rfloor_k := \{n \in \mathbb{N} \,|\, n - \frac{1}{k+1} < x < n + 1\} \cup \{0 \,|\, x < 0\}$ for $x \in \mathbb{R}$ and $k \in \mathbb{N}$.

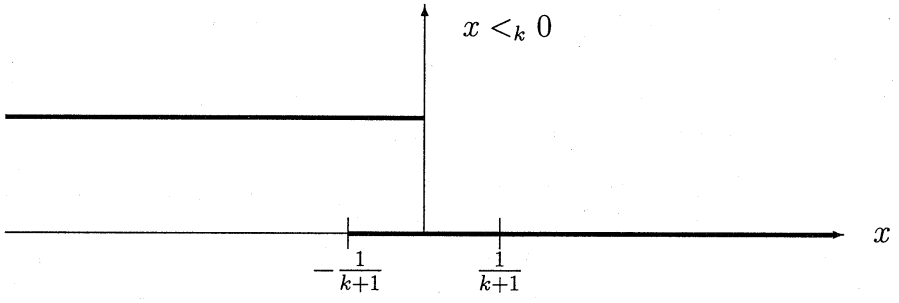

Figure 3: The finite precision test

It is an important observation that the finite precision test introduces an *indeterminism* into our feasible real RAM. More formally, a feasible real RAM computes a relation $R_M \subseteq X \times Y$ where $(x, y) \in R_M$ if there is some computation path on input x with output y. Here, X, Y denote finite products of $\mathbb{N}$ and $\mathbb{R}$. But in contrast to the kind of nondeterminism which is used in complexity theory, our indeterminism can be realized on physical machines. Indeterminism and nondeterminism have in common that a single input does potentially lead to several computation paths. While nondeterminism means that only some computation paths have to lead to a "valid result", indeterminism means that all computation paths have to lead to a "valid result". Thus, nondeterminism comes with an existential quantification and indeterminism with an universal quantification in the definition of semantics. To make this more precise we define our approximative semantics.

Definition 3.1 (Approximative semantics) Let M be a feasible real RAM with input space $X \times \mathbb{N}$ and output space Y and let $f :\subseteq X \to Y$, $t :\subseteq X \times \mathbb{N} \to \mathbb{N}$ be functions. Then M is said to *approximate* f in *time* t if $dom(f) \times \mathbb{N} \subseteq \mathrm{dom}(R_M)$ and

(1) $d(f(x), y) < 2^{-n}$ for all $(x, n) \in \mathrm{dom}(f) \times \mathbb{N}$ and $y \in R_M(x, n)$,

(2) $t_M(x, n) \leq t(x, n)$ for all $(x, n) \in \mathrm{dom}(f) \times \mathbb{N}$.

Here d denotes the maximum metric on Y, where Y is a finite product of the spaces $\mathbb{N}$, $\mathbb{R}$ which are equipped with the discrete metric, Euclidean metric, respectively. Furthermore, $t_M :\subseteq X \times \mathbb{N} \to \mathbb{N}$ denotes the *logarithmic time complexity* of the feasible real RAM M which can be defined by

$$t_M(x, n) := \max_{\text{comp. paths on } (x,n)} \{\sum_{op} \mathrm{cost}(op)\}.$$

for $(x, n) \in \mathrm{dom}(R_M)$ where the last sum is over all operations op in a computation path on input (x, n) and the maximum is over all computation paths on input (x, n). Thus, the logarithmic time complexity measure charges each operation with costs depending on the size of the operands. Now we are prepared to define time complexity classes for feasible real RAMs. For $t :\subseteq X \times \mathbb{N} \to \mathbb{N}$ we define

$$\mathrm{TIME}_{\mathrm{RAM}}(t) := \{f :\subseteq X \to Y \mid Y \text{ is a finite product of } \mathbb{N} \text{ and } \mathbb{R} \text{ and there is a RAM } M \text{ approximating } f \text{ in time } \mathcal{O}(t)\}.$$

Our main result in this section will compare these time complexity classes with the corresponding time complexity classes for Turing machines. We recall the fact that in computable analysis the time complexity of a function measures the number of Turing machine steps which are used in order to produce the result with precision 2^{-n} (cf. Ko [Ko91], Müller [Mül86], Weihrauch [Wei00]). Thus, a function $f :\subseteq X \to Y$ is in $\mathrm{TIME}_{\mathrm{TM}}(t)$ if and only if there is a Turing machine M which computes f with respect to the binary signed-digit representation and which produces an approximation of $f(x)$ with precision 2^{-n} for each input sequence p which represents x in at most $t(x,n)$ steps. Precise definitions can be found in [BH98]. Now our main result can be stated as follows.

Theorem 3.2 (Feasible real RAM) *For regular time bounds $t :\subseteq X \times \mathbb{N} \to \mathbb{N}$ the following inclusion holds:* $\mathrm{TIME}_{\mathrm{TM}}(t) \subseteq \mathrm{TIME}_{\mathrm{RAM}}(t) \subseteq \mathrm{TIME}_{\mathrm{TM}}(t^2 \cdot \log(t) \cdot \log\log(t))$.

Here a time bound $t :\subseteq X \times \mathbb{N} \to \mathbb{N}$ is called *regular*, if $\ell(x)+k+t(x,k+1) \subseteq \mathcal{O}(t(x,k))$ and $t(x,k) \geq 2$. This condition is fulfilled by all time bounds of practical interest. In other words our result states that feasible real RAMs are a polynomially realistic model of computability and complexity on the real numbers (compared to the Turing machine based model of computable analysis). One of the surprising parts of the result is that is suffices to use the logarithmic time complexity measure which only counts the sizes of the operands but which does not measure the precision of the local RAM operations which is necessary to compute the result.

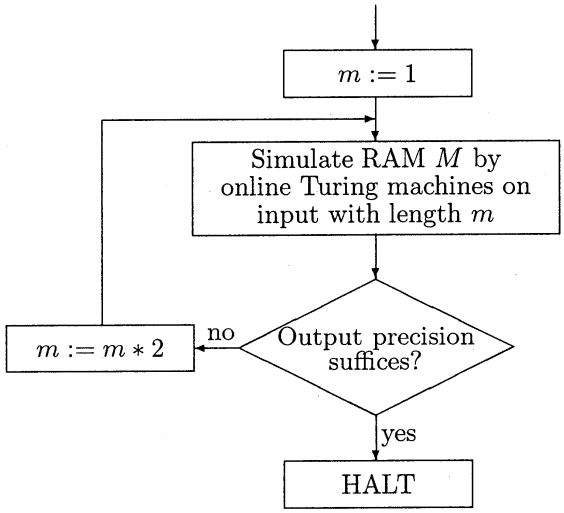

Figure 4: The simulation of a RAM M by a TM

The proof justifies this by the fact that the basic arithmetic operations are *online computable* in polynomial time, i.e. there are polynomial-time Turing machine programs for these operations such that the output precision is equal to the input precision minus a certain fixed delay depending on the input size. This result can be extended to other

larger classes of (smooth) basic operations like the exponential function or trigonometric functions (cf. [BH98]).

The main part of the proof which shows that feasible real RAMs can be simulated by Turing machines is based on the idea to start the simulation with a fixed input precision m and to restart it again and again with a doubling of the input precision until the output precision suffices. Because of regularity of the time bounds this doubling of precision will not effect the time complexity substantially. The flowchart in Figure 4 illustrates the idea.

4 Theory of continuous data structures

In the previous section we have seen how an abstract description of the model of computability of computable analysis can be developed, which is realistic with respect to computability and complexity. From a certain point of view we can interpret this result such that we have found an efficient data structure for real and natural numbers. This data structure can be summarized by the following table (where we have omitted operations which are not necessary from the point of view of computability).

$\mathbb{N}$	**Naturals**	$\{0,1,2,...\}$, discrete topology
0	constant	0
n	identity	$\mathrm{id} : \mathbb{N} \to \mathbb{N}, n \mapsto n$
$n+1$	successor function	$s : \mathbb{N} \to \mathbb{N}, n \mapsto n+1$

$\mathbb{R}$	**Reals**	computable real numbers, Euclidean topology
0	constant	0
1	constant	1
$x+y$	addition	$\mathbb{R} \times \mathbb{R} \to \mathbb{R}, (x,y) \mapsto x+y$
$-x$	negation	$\mathbb{R} \to \mathbb{R}, x \mapsto -x$
$x \cdot y$	multiplication	$\mathbb{R} \times \mathbb{R} \to \mathbb{R}, (x,y) \mapsto x \cdot y$
$1/x$	inversion	$\subseteq \mathbb{R} \to \mathbb{R}, x \mapsto 1/x$
$\lim_{n\to\infty} x_n$	limit	$\mathrm{Lim} :\subseteq \mathbb{R}^{\mathbb{N}} \to \mathbb{R}, (x_n)_{n\in\mathbb{N}} \mapsto \lim_{n\to\infty} x_n$
		$\mathrm{dom}(\mathrm{Lim}) := \{(x_n)_{n\in\mathbb{N}} : (\forall n > k)\ \lvert x_n - x_k\rvert \leq 2^{-k}\}$
$x <_k y$	finite precision test	$<_k : \mathbb{R} \times \mathbb{R} \times \mathbb{N} \rightrightarrows \mathbb{N}$

Table 3: The structure of natural and real numbers

Of course, in the feasible real RAM the limit operation was not an explicit basic operation but it was hidden in the approximative semantics.

In several areas of computer science we do not only want to compute with real numbers, but we also want to compute with sets of real numbers and real number functions. In numerical analysis we want for instance transform a program for a continuous function $f : [0,1] \to \mathbb{R}$ into a program for its integral function $x \mapsto \int_0^x f(t)\ dt$. Or in CAD we want to compute with compact sets of real numbers and perform operations like the union operation.

Consequently, the question appears how we can find suitable data structures not only for the real numbers but also for other spaces like hyper and function spaces. We just want to mention that an answer to this question can be given by the theory of *perfect structures* [Bra99b]. These structures, taken as data structures for a suitable high-level programming language or computability model (like the feasible RAM model) allow to compute exactly the same operations as the Turing machine model together with a corresponding representation. Moreover, perfect data structures have the following nice property [Bra99c].

Theorem 4.1 (Stability Theorem) *Perfect structures characterize their own computability theory.*

Instead of defining the notion of a perfect structure formally, we just mention two further examples.

$\mathcal{C}[0,1]$	**Continuous functions**	computable functions, topology of uniform convergence
1	constant function	$\{()\} \to \mathcal{C}[0,1], () \mapsto \hat{1}$
		$\hat{1} : [0,1] \to \mathbb{R}, x \mapsto 1$
f	identity	$\mathcal{C}[0,1] \to \mathcal{C}[0,1], f \mapsto f$
$y \cdot f$	scalar product	$\mathbb{R} \times \mathcal{C}[0,1] \to \mathcal{C}[0,1], (y,f) \mapsto y \cdot f$
$f+g$	addition	$\mathcal{C}[0,1] \times \mathcal{C}[0,1] \to \mathcal{C}[0,1], (f,g) \mapsto f+g$
$f \cdot g$	multiplication	$\mathcal{C}[0,1] \times \mathcal{C}[0,1] \to \mathcal{C}[0,1], (f,g) \mapsto f \cdot g$
$\|\|f\|\|$	norm	$\|\|\ \|\| : \mathcal{C}[0,1] \to \mathbb{R}, f \mapsto \sup_{x \in [0,1]} \|f(x)\|$
Lim	limit	$\mathrm{Lim} :\subseteq \mathcal{C}[0,1]^{\mathbb{N}} \to \mathcal{C}[0,1], (f_n)_{n\in\mathbb{N}} \mapsto \lim_{n\to\infty} f_n$
		$\mathrm{dom}(\mathrm{Lim}) := \{(f_n)_{n\in\mathbb{N}} : (\forall n > k) \|\|f_n - f_k\|\| \leq 2^{-k}\}$

Table 4: The structure of continuous functions

If we add to our data structure of the natural and real numbers the data structure for the space $\mathcal{C}[0,1]$ of continuous functions from Table 4 and for the space $\mathcal{K}(\mathbb{R}^n)$ of

non-empty compact subsets of $\mathbb{R}^n$ the data structure from Table 5, then we obtain again a perfect structure.

$\mathcal{K}(\mathbb{R}^n)$	**Compact subsets**	recursive compact sets, Vietoris topology
$\{x\}$	injection	$\mathbb{R}^n \to \mathcal{K}(\mathbb{R}^n), x \mapsto \{x\}$
A	identity	$\mathcal{K}(\mathbb{R}^n) \to \mathcal{K}(\mathbb{R}^n), A \mapsto A$
$A \cup B$	union	$\mathcal{K}(\mathbb{R}^n) \times \mathcal{K}(\mathbb{R}^n) \to \mathcal{K}(\mathbb{R}^n), (A,B) \mapsto A \cup B$
$d_{\mathcal{K}}$	Hausdorff metric	$d_{\mathcal{K}} : \mathcal{K}(\mathbb{R}^n) \times \mathcal{K}(\mathbb{R}^n) \to \mathbb{R}$, $(A,B) \mapsto \max\left\{\sup_{a\in A}\inf_{b\in B} d(a,b), \sup_{b\in B}\inf_{a\in A} d(a,b)\right\}$
Lim	limit	$\mathrm{Lim} :\subseteq \mathcal{K}(\mathbb{R}^n)^{\mathbb{N}} \to \mathcal{K}(\mathbb{R}^n), (A_n)_{n\in\mathbb{N}} \mapsto \lim_{n\to\infty} A_n$, $\mathrm{dom}(\mathrm{Lim}) := \{(A_n) : (\forall n > k)\ d_{\mathcal{K}}(A_n, A_k) \leq 2^{-k}\}$

Table 5: The structure of non-empty compact sets

Thus, we have here two possible data structures for computations with objects like compact sets and continuous functions. Of course, these structures represent only examples of data structures and there are other structures which correspond to different topologies and computability notions on the same spaces. The tables mention the underlying topologies and the corresponding sets of computable points.

Moreover, the theory of perfect structures can answer the question which asks for reasonable data structures only with respect to computability and nothing is said about complexity. For complexity questions, a comprehensive and general theory of computational complexity of metric spaces is still missing.

5 Conclusion

In the introduction we have described the role of the computability model in the cycle of real number programming. As the cardinal problem of the real number model we have singled out the discontinuity of the comparisons and equality tests. On the on hand, this causes a discontinuous description of the problem under consideration and, on the other hand, this makes a precise implementation on physical computers impossible.

In Section 2 we have presented some basic results of computable analysis. Especially, it has been described precisely which computations on the real numbers can be performed realistically on physical computers. One central result states that only continuous operations can be performed.

In Section 3 we have presented the feasible real RAM model which is an approach to describe the results of computable analysis in terms which are as close as possible to the real number model.

In Section 4 we have shown, at least on the level of computability, how these ideas can be transferred to computations with other objects like compact subsets and continuous functions.

Finally, we propose to replace the original real number model in the cycle of real number programming by the feasible real RAM model. Such a substitution would have several advantages: on the one hand, real world problems, especially if they are inspired by physical questions, are typically continuous. These problems can be described pretty good by the continuous (but indeterministic!) feasible real RAM model. On the other hand and maybe more important, each correct algorithm, developed for the feasible real RAM model, can be implemented on physical computers without any instability or degeneracy problems!

References

[BH98] Vasco Brattka and Peter Hertling. Feasible real random access machines. *Journal of Complexity*, 14(4):490–526, 1998.

[BMS94] Christoph Burnikel, Kurt Mehlhorn, and Stefan Schirra. On degeneracy in geometric computations. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (Arlington, VA, 1994)*, pages 16–23, New York, 1994. ACM.

[Bra99a] Vasco Brattka. The emperor's new recursiveness. preprint, 1999.

[Bra99b] Vasco Brattka. Recursive and computable operations over topological structures. Informatik Berichte 255, FernUniversität Hagen, Fachbereich Informatik, Hagen, July 1999. Dissertation.

[Bra99c] Vasco Brattka. A stability theorem for recursive analysis. In Cristian S. Calude and Michael J. Dinneen, editors, *Combinatorics, Computation & Logic*, Discrete Mathematics and Theoretical Computer Science, pages 144–158, Singapore, 1999. Springer. Proceedings of DMTCS'99 and CATS'99, Auckland, New Zealand, January 1999.

[BSS89] Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, July 1989.

[Grz57] Andrzej Grzegorczyk. On the definitions of computable real continuous functions. *Fundamenta Mathematicae*, 44:61–71, 1957.

[Hau73] Jürgen Hauck. Berechenbare reelle Funktionen. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 19:121–140, 1973.

[HW94] Peter Hertling and Klaus Weihrauch. Levels of degeneracy and exact lower complexity bounds for geometric algorithms. In *Proceedings of the Sixth Canadian Conference on Computational Geometry*, pages 237–242. University of Saskatchewan, 1994. Saskatoon, Saskatchewan, August 2-6, 1994.

[Ko91] Ker-I Ko. *Complexity Theory of Real Functions.* Progress in Theoretical Computer Science. Birkhäuser, Boston, 1991.

[KW84] Christoph Kreitz and Klaus Weihrauch. A unified approach to constructive and recursive analysis. In M.M. Richter, E. Börger, W. Oberschelp, B. Schinzel, and W. Thomas, editors, *Computation and Proof Theory*, volume 1104 of *Lecture Notes in Mathematics*, pages 259–278, Berlin, 1984. Springer. Proceedings of the Logic Colloquium, Aachen, July 18-23, 1983, Part II.

[Lac55] Daniel Lacombe. Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles I. *Comptes Rendus Académie des Sciences Paris*, 240:2478–2480, June 1955. Théorie des fonctions.

[Mül86] Norbert Th. Müller. Computational complexity of real functions and real numbers. Informatik Berichte 59, FernUniversität Hagen, Hagen, June 1986.

[PER89] Marian B. Pour-El and J. Ian Richards. *Computability in Analysis and Physics.* Perspectives in Mathematical Logic. Springer, Berlin, 1989.

[PS85] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry.* Texts and Monographs in Computer Science. Springer, New York, 1985.

[Tur36] Alan M. Turing. On computable numbers, with an application to the "Entscheidungsproblem". *Proceedings of the London Mathematical Society*, 42(2):230–265, 1936.

[Tur37] Alan M. Turing. On computable numbers, with an application to the "Entscheidungsproblem". A correction. *Proceedings of the London Mathematical Society*, 43(2):544–546, 1937.

[TWW88] Joseph F. Traub, G.W. Wasilkowski, and H. Woźniakowski. *Information-Based Complexity.* Computer Science and Scientific Computing. Academic Press, New York, 1988.

[Wei98] Klaus Weihrauch. A refined model of computation for continuous problems. *Journal of Complexity*, 14:102–121, 1998.

[Wei00] Klaus Weihrauch. *An Introduction to Computable Analysis.* Springer, Berlin, 2000. (to appear).