Basic Research in Computer Science

A Two-Layer Approach to the Computability and Complexity of Real Numbers

Branimir Lambov

BRICS Report Series **RS-03-50**

ISSN 0909-0878 **December 2003**

See back inner page for a list of recent BRICS Report Series publications.
Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/03/50/`

A two-layer approach to the computability and complexity of real functions

Branimir Lambov

BRICS*
Department of Computer Science
University of Aarhus
DK-8000 Aarhus C
Denmark
barnie@brics.dk
December, 2003

AMS Classification: 03D65, 03D15, 68Q15, 68Q05, 03D80

Abstract

We present a new approach to computability of real numbers in which real functions have type-1 representations, which also includes the ability to reason about the complexity of real numbers and functions. We discuss how this allows efficient implementations of exact real numbers and also present a new real number system that is based on it.

1 Introduction

Most theoretical approaches to the computability of real functions rely on higher-type computations, taking a description of a real number into a computation of another description. In practice, creating and maintaining such descriptions is a very complicated process, requiring extra storage along with time-consuming memory management tools such as garbage collection and, if not enough care is applied, also introducing external complexity that may even lead to changing the complexity class of the problem.

*Basic Research in Computer Science (`www.brics.dk`), funded by the Danish National Research Foundation.

For example, to compute simply $a * a$, where a is a real number described as some function computing its approximations, and $*$ is a type-2 function that computes multiplication on the reals, the straightforward implementation would require the computation of approximations to a twice. Then, an efficient implementation of a^n where $n \in \mathbf{N}$ using $*$, which on ground types would have complexity of operations $O(\log n)$, would inevitably end up being of complexity $O(n)$, because every multiplication would require computing its arguments, leading to at least n requests to compute approximations to a.

The problem in this example can be circumvented by introducing approximation caching. Unfortunately, this leads to other issues and the process of circumventing problems can go on indefinitely.

This paper tries to address this problem from the roots, creating a firm theory on which implementations without such efficiency problems can be devised. The theory presented is a new model of computation, based on representing real functions as objects operating on *partial approximations* instead of proper real numbers. The novelty of this model is in the introduced second layer that adds the ability to reason about complexity, at the same time separating these concerns from the objects that carry out the computation. No other type-1 model so far has been able to define complexity, and ours manages to do it in a way that is compatible with existing complexity measures for computable analysis.

In the next section we will present the two-layer approach, prove its properties, and compare it to existing models. Following this, we will discuss the usefulness of the ideas presented, along with a short introduction to an actual implementation and the conclusions it allows us to draw. At the end we will take a look at the future plans for this research.

2 Model

2.1 Established notions

First we will introduce two widely accepted notions of computability on real numbers and functions on real numbers, which we will use to justify that the definitions in the model coincide with the established ones.

Let $\mathbf{B}$ be a computable base for $\mathbf{R}$ that contains the dyadic numbers, and $\mathbf{B}_\infty = \{x : x \in \mathbf{B} \wedge x > 0\} \cup \{\infty\}$ together with operations that respect ∞. To allow us to talk about feasibility, we will use $\mathrm{lth}(n) = \lfloor \log_2 n \rfloor + 1$ in the exponents instead of simply n.

Let there exist computable and even poly-time encodings of the elements of $\mathbf{B}$ and $\mathbf{B}_\infty$ and of the pairings $\mathbf{B} \times \mathbf{B}_\infty$. In some of the proofs we will also need certain basic properties of the encodings:

- $a_1 \leq a_2 \wedge b_1 \leq b_2 < \infty \rightarrow \langle (a_1, b_1) \rangle \leq \langle (a_2, b_2) \rangle$
- $\mathrm{lth}(\langle (a, b) \rangle)$ is polynomial in $\max(\mathrm{lth}(a), \mathrm{lth}(b))$
- there exists a function $rat(n, 2^d)$ that selects a code for the rational number $n2^{-d}$, such that whenever a, b, c, d are positive integers, $a \leq c \wedge b \leq d \rightarrow rat(a, 2^b) \leq rat(c, 2^d)$
- the absolute value operator on the rational codes is such that $\langle b \rangle \leq \langle |b| \rangle$ for any rational b

These properties are satisfied by e.g. the Cantor pairing and the encoding of rational numbers q as pairs (n, d), such that

$$q = (-1)^n \frac{\lfloor (n+1)/2 \rfloor}{d}.$$

Definition 2.1 *A Cauchy function representation (CF-representation) of a real number α is a function $a : \mathbf{N} \rightarrow \mathbf{B}$, such that $\forall n \in \mathbf{N} \left(|a(n) - \alpha| < 2^{-\mathrm{lth}(n)} \right)$*

Definition 2.2 *A Cauchy function representation of a partial function $\phi : \mathbf{R} \rightarrow \mathbf{R}$ is a partial functional $\Phi : (\mathbf{N} \rightarrow \mathbf{B}) \times \mathbf{N} \rightarrow \mathbf{B}$, such that*

$$\forall \alpha \in \mathrm{dom}\ \phi, \forall a - \mathrm{CF} - \mathrm{representations\ of}\ \alpha$$
$$\forall n \in \mathbf{N} \left((a, n) \in \mathrm{dom}\ \Phi \wedge |\Phi(a, n) - \phi(\alpha)| < 2^{-\mathrm{lth}(n)} \right)$$

Definition 2.3 *A real number or a real function is computable in a class C of computable functions, resp. functionals, iff there exists a representation in C for it in the sense of Definitions 2.1 or 2.2 respectively.*

2.2 Partial approximation representations of real numbers

The inefficiency in these definitions comes from the higher type of the function object Φ in Definition 2.2. To alleviate this, we need to transfer the precision information to the type-0 level. Our approach to this is similar to the domain theoretic and the interval arithmetic ideas, and uses a basic object called a partial approximation.

Definition 2.4 *A* partial approximation *to a real number α is a pair (v, e) of type $\mathbf{B} \times \mathbf{B}_\infty$, such that $|v - \alpha| < e$. We will denote the class of partial approximations to α with $\mathbf{A}_\alpha$, and the class of partial approximations to any real with $\mathbf{A}_\mathbf{R} = \cup_{\alpha \in \mathbf{R}} \mathbf{A}_\alpha$. If $a \in \mathbf{A}_\mathbf{R}$ we will use a_v, a_e to denote respectively the value and error in a.*

Definition 2.5 *A* partial approximation representation, p.a.r., *of a real number α is a function $A : \mathbf{N} \to \mathbf{A}_\alpha$, for which $\forall k \exists n((A(n))_e \leq 2^{-k})$.*

If a real number is computable, then it certainly has a computable p.a.r.: if B is a representation of α, then $\lambda n.(B(n), 2^{-\mathrm{lth}(n)})$ is one of its p.a.r.'s. Conversely, if a is a p.a.r. of α, then $\lambda k.A(\mu n[A(n)_e \leq 2^{-\mathrm{lth}(k)}])_v$ is a valid CF-representation for it.

This equivalence does not hold for restrictions of the notion of computablily. Moreover, it is possible to define all computable reals using p.a.r.'s in subrecursive classes such as primitive recursive, elementary or poly-time functions. For a proof of this, see [12].

In order to be able to speak about different complexity classes of real numbers, a similar equivalence must be available. This gives rise to the following definitions and equivalence property:

Definition 2.6 *A* modulus *for a p.a.r. A of a real number α is a function $m : \mathbf{N} \to \mathbf{N}$, such that for all k, $(A(m(k)))_e \leq 2^{-\mathrm{lth}(k)}$.*

Definition 2.7 *We will say that a real number is p.a.r.-computable in a given class C of computable functions, if there exist both a p.a.r. and a modulus for it in C.*

Theorem 2.8 *A real number is computable in a subrecursive class C that contains the poly-time functions and is closed under composition if and only if it is p.a.r.-computable in C.*

Proof Take $A := \lambda n.(B(n), 2^{-\mathrm{lth}(n)})$ and $m := \lambda n.n$, or for the other direction $B := \lambda k.A(m(k))$. •

On the level of feasible functions, poly-time p.a.r. computability coincides with Ko's notion of poly-time computable real numbers [8] (Ko speaks about numbers given in unary notation, which is equivalent to the $\mathrm{lth}(n)$ parameter used in our definitions).

2.3 Partial approximation representations of real functions

For real functions, we want to have objects that operate on partial approximations instead of the full representations. They will have to convert approximations to an input to approximations to the result of the application of the function, and also we need to require that the precision of the output approximations gets arbitrarily good as the precision of the input increases. In other words,

Definition 2.9 *A* partial approximation representation *of a partial function* $\phi : \mathbf{R} \to \mathbf{R}$ *is a partial function* $F : \mathbf{A_R} \to \mathbf{A_R}$*, such that for any choice of* $\alpha \in \text{dom}\,\phi$ *and a partial approximation representation* A *of* α*,* $\lambda n.F(A(n))$ *is a partial approximation representation of* $\phi(\alpha)$*.*

Remark 2.10 *This definition implies* $a \in \mathbf{A}_\alpha \to F(a) \in \mathbf{A}_{\phi(\alpha)}$ *for* $\alpha \in \text{dom}\,\phi$*, because for any p.a.r.* A *we can create* $B(n) := \overline{sg}(n) \cdot a + sg(n) \cdot A(n-1)$*, which is another p.a.r. of* α *that has* $B(0) = a$*, hence* $F(a)$ *has to be a partial approximation to* $\phi(\alpha)$*.*

Remark 2.11 *Unlike the domain theoretic and interval arithmetic approaches, we do not require the image interval to be described accurately. Our requirement is only that we are able to provide a superset of it, and this superset gets smaller as the input interval gets smaller.*

2.3.1 Computability

We have severely restricted the information to which the function object has access; nevertheless, the following theorem proves we have not restricted the class of real functions that are computable:

Theorem 2.12 *A partial function* $\phi : \mathbf{R} \to \mathbf{R}$ *is computable iff it has a computable p.a.r.*

Proof ($\leftarrow$) If we have a p.a.r. F of a function ϕ, and $\alpha \in \text{dom}\,\phi$, then the functional

$$\Phi(B,n) := \text{let } \left(m = \mu p \left[(F(B(p), 2^{-\text{lth}(p)})_e \leq 2^{-\text{lth}(n)}\right]\right) \text{ in } F(B(m), 2^{-\text{lth}(m)})$$

is total in n for any CF-representation B of α since from Definitions 2.9 and 2.5 the minimization will always stop, and Definition 2.4 together with Remark 2.10 ensures $|\Phi(a,n) - \phi(\alpha)| < 2^{-\text{lth}(n)}$. •

Proof ($\rightarrow$) We have a fixed $\alpha \in \text{dom}\,\phi$, and a CF-representation Φ for ϕ.

For any $a \in \mathbf{A}_\alpha$ with $a_e < 1$, we can effectively find the largest natural number m with the property $2^m a_e < 1$. If $a_e \geq 1$, we take $m = 0$. Define the function

$$b := \lambda n.2^{-\text{lth}(n)} \lfloor 2^{\text{lth}(n)} a_v + 1/2 \rfloor. \tag{1}$$

For $0 \leq \text{lth}(n) < m$ we have that

$$|b(n) - \alpha| \leq |a_v - \alpha| + 2^{-(\text{lth}(n)+1)} \leq 2^{-m} + 2^{-(\text{lth}(n)+1)} \leq 2^{-\text{lth}(n)}$$

Using the fact that exceptions are computable and that given the code of a computable functional Φ, we can construct a parallel one $\Phi^\dagger$ that honors a new exception x, we can effectively create a function

$$b\lceil m := \lambda n. \begin{cases} b(n), & \text{if } n < m \\ \mathbf{raise}\, x, & \text{otherwise} \end{cases}$$

and then define

$$\Phi^\ddagger(B, n) := \mathbf{try}\ \Phi^\dagger(B\lceil m, n) + 1\ \mathbf{catch}(x)\ 0. \tag{2}$$

(i.e. $\Phi^\ddagger$ will return $\Phi(b, n) + 1$ if b restricted to length m was sufficient to compute it, and 0 otherwise)

We will now prove that the function

$$F(a) := \begin{cases} (0, \infty), & \text{if } l = 0 \\ (\Phi^\ddagger(b, l-1) - 1, 2^{-\mathrm{lth}(l-1)}), & \text{otherwise} \end{cases}$$

for

$$l = \mu p \leq m \left[\Phi^\ddagger(b, p) = 0\right] \tag{3}$$

is the required p.a.r. of ϕ. To do this, we need to prove that $G = \lambda n.F(A(n))$ is a p.a.r. of $\phi(\alpha)$ for any p.a.r. A of α.

The first condition, $F(a) \in \mathbf{A}_{\phi(\alpha)}$ for any $a \in \mathbf{A}_\alpha$, follows from the requirement for Φ and the fact that there is a CF-representation for α that starts with $b(0), b(1), \ldots, b(m-1)$.

For the second condition, we need to prove the existence of 2^{-k}-approximations to $\phi(\alpha)$ among $G(n)$ for any k. The sequence defined by

$$c := \lambda n.2^{-\mathrm{lth}(n)} \lfloor 2^{\mathrm{lth}(n)} \alpha + 1/2 \rfloor$$

is a proper CF-name for α. If α is not a dyadic number, then for an arbitrary n, $|\alpha - c(n)| < 2^{-(n+1)}$. There exists q depending on n, such that $|\alpha - c(n)| \leq 2^{-n}(1/2 - 2^{-(q-n)})$, and for all partial approximations a with $a_e < 2^{-q}$ we have $2^i |a_v - c(i)| < 1/2$ for all $0 \leq i \leq n$. But this implies that the sequence obtained by (1) coincides with c on the first $n+1$ elements.

Now, since Φ would look at finitely many elements of c to produce a value with any precision 2^{-k}, using that count in the procedure described above, we can come up with a q supplying a long enough sequence. Combining this with a requirement that the minimization (3) reaches the target precision, we have $(F(a))_e \leq 2^{-k}$ for all a's with $a_e \leq 2^{-\max(q,k)}$, and since A has arbitrarily close approximations, this can be satisfied for $a = A(n)$ for some n.

If α is a dyadic number, i.e. $\exists c, n(c(n) = \alpha)$, then there are only finitely many variations of b that can exists, because they have to coincide after the first $n + 1$ positions. Then there exists a maximum m for the number of lookups Φ can make to any of these b's in order to get a 2^{-k}-precise result. Hence $a_e \leq 2^{-\max(m,k)}$ suffices to get the required precision for $F(a)$. •

Remark 2.13 *The proof of the existence of arbitrarily close approximations in this form is ineffective. There also exists a construction for which this proof can be carried out effectively (by making the choice of b non-deterministic and using a representation generated by A instead of c). The presented proof, however, is much more easily adjustable to the restricted conditions in which it will be used later.*

As well as in the case of real numbers, this equivalence does not hold for subclasses of the type-2 computable functions. To define all computable functions, it suffices to use severely restricted type-1 computability subclasses:

Theorem 2.14 *A partial real function is computable iff it has a p.a.r. in any subrecursive class C that contains the poly-time functions.*

Proof ($\rightarrow$) It suffices to change the definition of Φ^- to a version bounded in execution time:

$$\Phi^-(B, n) := \mathbf{try}\ \Phi^m(B\lceil m, n) + 1\ \mathbf{catch}(x)\ 0.$$

where by Φ^m we denote $\Phi^\dagger$ executed for m steps, which is a basic feasible functional (BFF, [1]).

Since m is of the order of $\mathrm{lth}(a)$ for the encoding of a it is possible to do all required steps in time polynomial to $\mathrm{lth}(a)$. The proof of the existence of good appoximations can be carried out here as well, the only difference being the need to satisfy a condition in the form $a_e \leq 2^{-max(q,k,s)}$ for s being the number of steps it takes for Φ to complete its evaluation on b of length q.

The p.a.r. is type-1 basic feasible, therefore it is poly-time by a known property of the BFF [2]. •

Proof ($\leftarrow$) Follows from the previous theorem. •

2.3.2 Type-2 complexity

Again taking the p.a.r. of a real function we lose all complexity information about that function. To talk about complexity classes again, we define

Definition 2.15 *A* modulus *for a p.a.r. F of a partial real function ϕ is a partial functional $M : (\mathbf{A_R} \to \mathbf{A_R}) \times (\mathbf{N} \to \mathbf{N}) \times \mathbf{N} \longrightarrow \mathbf{N}$, such that for all $\alpha \in \operatorname{dom} \phi$, p.a.r. A of α, moduli m for A,*

$$\forall k((F(A(M(A, m, k))))_e \leq 2^{-\mathrm{lth}(k)}). \tag{4}$$

Note that even though the actual function object is a type-1 object, we now introduce a type-2 operation to characterize it. However, the strength comes from the separation of these two objects: to have a e.g. a feasible real function you do not have to have a feasible type-2 object, but only need to prove that it exists. Moreover, if a CF-representation of a function needs extra information to be in a certain class (e.g. division needs evidence that the denominator is non-zero to be primitive recursive), it will in general only be needed for the modulus.

Definition 2.16 *We will say that a real function is p.a.r.-computable in a given class C of computable type-2 functionals, if both a computable p.a.r. and its modulus can be found in C.*

Theorem 2.17 *If a function is p.a.r.-computable in a given class C that contains BFF and is closed under functional substitution, then it is computable in the same class.*

Proof For $\phi : \mathbf{R} \to \mathbf{R}$, $\alpha \in \operatorname{dom} \phi$, F- p.a.r. of ϕ, M-modulus for F, and B - CF-representation of α, take

$$\Phi(B, n) := (F(A(M(A, \lambda p.p, n))))_v$$

where

$$A := \lambda p.(B(p), 2^{-\mathrm{lth}(p)}).$$

A is a p.a.r. for α with a modulus $\lambda p.p$, and hence from M being a modulus to F, we have $|\Phi(B, n) - \phi(\alpha)| < 2^{-\mathrm{lth}(n)}$. Φ is a basic feasible functional relative to F and M, therefore it is in C. •

Theorem 2.18 *If a partial function $\phi : \mathbf{R} \to \mathbf{R}$ is computable, then it is p.a.r.-computable.*

Proof We've already proved in Theorem 2.12 that there exists a computable p.a.r. to every computable real function. If it is F, then

$$M(A, m, n) := \mu p[F(A(p))_e \leq 2^{-\mathrm{lth}(n)}]$$

is a modulus for F. •

This modulus does not even use the modulus for the real number. This is true, because in the presence of minimization brute force search makes the moduli redundant.

This is not the case for restricted complexity classes. To prove the equivalence between p.a.r. and CF-computability on some of them, we will introduce the notion of *majorizability*.

Definition 2.19 (W.A. Howard [5]) *We define $x^* \ maj_\rho \ x$ for a finite type ρ by induction on the type:*

$$x^* \ maj_0 \ x := x^* \geq x,$$

$$x^* \ maj_{\tau\to\rho} \ x := \forall y^*, y \left(y^* \ maj_\tau \ y \to x^* y^* \ maj_\rho \ xy \right).$$

We will say that a subrecursive class C is majorizable, if for every function f in C there exists $f^* \in C$ with $f^* \ maj \ f$, where the majorization operator is of the appropriate type.

Lemma 2.20 *The class of primitive recursive type-2 functionals is majorizable.*

This lemma is a corollary to the fact that the class of the primitive recursive functionals of finite type (PR^ω) are majorizeable, proved in e.g. [9]. We will use the same technique to prove

Lemma 2.21 *The class of basic feasible type-2 functionals (BFF) is majorizable.*

which is a corollary to

Lemma 2.22 *The class of basic feasible functionals of finite type (BFF^ω) is majorizable.*

Proof We will use the fact [2] that every functional in BFF^ω can be written as a term which only contains constants 0_ρ, variables y_ρ, poly-time functions, $\Sigma_{\delta,\rho,\tau}$, $\Pi_{\rho,\tau}$ and bounded recursion on notation R_{bn}:

$$R_{bn}(x,y,g,h) = \begin{cases} y, & \text{if } x = 0 \\ \min\left(g(x, R_{bn}(\lfloor x/2 \rfloor, y, g, h), h(x)\right), & \text{otherwise} \end{cases}$$

0_ρ, y_ρ, $\Sigma_{\delta,\rho,\tau}$ and $\Pi_{\rho,\tau}$ are self majorizable, and for every poly-time function f there exists a polynomial p with coefficients among the natural numbers, such that $f(\underline{x}) \leq 2^{p(\mathrm{lth}(\underline{x}))} = f^*(\underline{x})$. But the right hand side of this inequality is a polytime function for which $\forall \underline{x} \forall \underline{y} \leq \underline{x} \left(f^*(\underline{x}) \geq f^*(\underline{y}) \geq f(\underline{y}) \right)$, i.e. $f^* \ maj_1 \ f$. Define

$$R^*_{bn}(x,y,g,h) := h(x)$$

If $x^*, y^* \geq x, y$, $g^*\ maj_{0\to 0\to 0}\ g$ and $h^*\ maj_1\ h$

$$R^*_{bn}(x^*, y^*, g^*, h^*) = h^*(x^*) \geq h(x) \geq R_{bn}(x, y, g, h),$$

which proves $R^*_{bn}\ maj_{0\to 0\to (0\to 0\to 0)\to 1\to 0}\ R_{bn}$.

Now the result follows from the fact that $t^*\ maj_{\rho\to\tau}\ t \wedge s^*\ maj_\rho\ s$ implies $t^*s^*\ maj_\tau\ ts$. •

To prove equivalence of the computability classes, we will also need

Lemma 2.23 *The following variation of the function b created in (1) along the course of evaluation of F on a p.a.r. A*

$$b(n) := rat(\lfloor 2^{\mathrm{lth}(n)} a_v + 1/2 \rfloor, 2^{\mathrm{lth}(n)}) \tag{5}$$

can be majorized on partial approximations with error ≤ 1 if there is an upper bound for the absolute value of the real number described by A.

Proof Let a_0 be a rational number such that $|\alpha| < a_0$ for α being the real described by A. Then for any partial approximation a with $a_e \leq 1$ we have $|a_v| < a_0 + 1$ and therefore by the properties of the encoding

$$b^*(n) = rat(1 + \lfloor 2^{\mathrm{lth}(n)} a_0 + 1/2 \rfloor, 2^{\mathrm{lth}(n)}) \geq b(n)$$

and also, since when n is increased both the numerator and denominator will not decrease, we have $\forall k \leq n(b^*(n) \geq b^*(k) \geq b(k))$, which means $b^*\ maj_1\ b$. •

Theorem 2.24 *If a partial real function is computable in a majorizeable class of type-2 functionals that contains BFF and is closed under functional substitution, then it is p.a.r.-computable in that class.*

Proof We will use the proof of Theorem 2.12, substituting the definition of b with (5). All operations used in the generation of F can be done without leaving the class of Φ. Hence F is in the class. We now need to find a modulus for it.

In the class of Φ there exists a functional Ψ that does exactly the same job as Φ, but instead of returning the approximation it gives the largest k to which B was applied. Since the class contains this functional and is majorizable, it also contains a majorizer Ψ^* for it. The modulus for A gives us means to bound the absolute value of the real number described by it, therefore, with the previous lemma, in our class there is a function J that given A and m computes b^* which majorizes all functions b generated by good partial approximations.

Hence $\Psi^*(b^*, n) \geq \Psi(b, n)$ for all good b's, in particular for the one (call it b_0) generated by $a_0 = A(m(\Psi^*(b^*, n)))$, which means $\Phi^\dagger(b_0, n)$ will not raise an exception, and $F(a_0)$ will give a result with the required precision.

Hence $M(A, m, n) = m(\Psi^*(J(A, m), n))$ is a modulus for F. •

2.3.3 Real number complexity

In the previous subsection we found correspondence between complexities in this model and type-2 complexity. As this is not the complexity measure normally used for real functions, we also define notions which are more closely related to the latter by defining type-1 moduli on closed subsets of the domain:

Definition 2.25 *A* uniform modulus *on* $[a,b] \subseteq \operatorname{dom}\phi$ *of a p.a.r.* F *of a real function* ϕ *is a function* $U : \mathbf{N} \to \mathbf{N}$*, such that*

$$\forall \alpha \in [a,b] \forall A - p.a.r.\ of\ \alpha \forall k \forall n (A(n)_e \leq U(k) \to (F(A(n)))_e \leq 2^{-\operatorname{lth}(k)})$$

Theorem 2.26 *A partial real function* ϕ *is computable in a majorizable class of type-2 functionals on* $[a,b] \subseteq \operatorname{dom}\phi$ *if and only if it has a p.a.r. and a* uniform modulus *in the same class.*

Proof ($\to$) Use a and b to find an upper bound for the absolute value of α, then apply the same reasoning as in the previous proof. •

Proof ($\leftarrow$) $M(A,m,k) = m(U(k))$ is a modulus for all A's representing reals in the interval, thus ϕ is p.a.r.-computable in the class. •

With this definition we're back at the type-1 level, and we also have a few important equivalences:

Corollary 2.27 *A partial real function* ϕ *is primitive recursive on* $[a,b] \subseteq \operatorname{dom}\phi$ *iff it has a primitive recursive p.a.r. and a primitive recursive uniform modulus on* $[a,b]$.

Corollary 2.28 *A partial real function* ϕ *is BFF-computable on* $[a,b] \subseteq \operatorname{dom}\phi$ *iff it has a poly-time p.a.r. and a poly-time uniform modulus on* $[a,b]$.

And this combined with the following theorem gives us equivalence with the established notion for feasible real functions.

Theorem 2.29 *A partial real function is poly-time computable on* $[a,b]$ *in the sense of Ko's defintion [8] iff it is BFF-computable.*

Proof A function is poly-time computable in Ko's sense iff there is an oracle Turing machine computing it in the dyadic representation, which runs in time polynomial to the precision given in 1^k notation.

Type-2 complexity theory says a functional is in BFF if and only if there exists an oracle Turing machine computing it running in time which is a second-order polynomial in the length of the inputs [6]. There exist representations of any real number that satisfy $\operatorname{lth}(B(k)) \leq p(\operatorname{lth}(k))$ for a polynomial p (using dyadic representations cut after the $\operatorname{lth}(k)$'th digit), and therefore a second order polynomial in $\operatorname{lth}(k), \operatorname{lth}(B)$ does not give more power than simply a polynomial in $\operatorname{lth}(k)$. •

3 An example: reciprocal of a real number

The task in this section will be to define a suitable p.a.r. of the function $1/\alpha : \mathbf{R} \to \mathbf{R}$ and to inspect its properties.

Theorem 3.1 *The poly-time function F defined as*

$$F(a) = \begin{cases} (\frac{1}{a_v}, \frac{a_e}{|a_v|(|a_v|-a_e)}), & \text{if } a_e < |a_v| \\ (0, \infty), & \text{if } a_e \geq |a_v| \end{cases} \tag{6}$$

is a p.a.r. of the reciprocal function on the reals.

Proof Given a fixed $\alpha \neq 0$, if a is a partial approximation to α with $a_e < a_v$, then

$$\left|\frac{1}{\alpha} - \frac{1}{a}\right| \leq \frac{1}{|a_v| - a_e} - \frac{1}{|a_v|} \leq \frac{|a_v| - (|a_v| - a_e)}{|a_v|(|a_v| - a_e)} = \frac{a_e}{|a_v|(|a_v| - a_e)},$$

which means F converts partial approximations to α into partial approximations to $1/\alpha$. If a_0 is a positive rational number smaller than $|\alpha|$, then to get $(F(a))_e \leq \varepsilon$, it is enough to supply a partial approximation with $a_e < a_0^2\varepsilon/2$ (assuming $a_0, \varepsilon \leq 1$).
•

It is a well known fact that the reciprocal function on the reals is not even primitive recursively computable. However, it is poly-time computable on every closed interval that does not contain 0. How does this translate to our framework?

Having a closed interval that does not contain 0 is equivalent to having witness information for the strict positivity of $|\alpha|$. But this extra information is enough to allow us to define

$$M(a_0, A, m, k) = m(a_0 \# a_0 \# k \# 2)$$

$$U(a_0, k) = 2^{-2(\mathrm{lth}(a_0))^2 \mathrm{lth}(k)}$$

(with # being the smash function, $x\#y = 2^{\mathrm{lth}(x)\mathrm{lth}(y)}$) which are, respectively, a modulus for F and an uniform modulus for F on the full real line. Therefore, F defined in (6) is uniformly linear in the error of the approximation a_e and the requested precision k, and quadratic in the value of the appriximation a_v and the witness a_0 on the full real interval.

Note that we needed the witness only to define and prove a property of the p.a.r. The representation itself is not changed by whether it can be found or not.

4 Comparison with existing models

During the presentation of the model we made some parallels with existing models. But how do we actually compare to them?

This model has the same expressive power as the type-2 model for Computable Analisys as treated for example in [13]. Important restricted variations and subclasses of the computable real functions and numbers can also be defined in our model with straightforward and natural definitions. The advantage of this model is in its ability to avoid type-2 objects and still define the same notions.

In the setting of feasible real functions, this model complies with Ko's definition of poly-time computability [8]. Moreover, our model is able to define feasibility of a representation of a function avoiding the use of an oracle Turing machine and any type-2 object at all using the definition of the uniform modulus for a p.a.r. It is an interesting question how it can be extended to arbitrary Polish spaces and what definition of feasibility on type-2 in general that would give.

The core of this model, the p.a.r.'s without their moduli, is very similar to the domain theoretic model for computability on the real line of Edalat and Sünderhauf [3]. The most significant distinction of ours is in dropping the requirement of accurately describing the image interval of a function's application on an interval, along with freedom to omit unnecessary data from the sequences describing real numbers. With this relaxation we have been able to define complexity in this model using restricted classes of real numbers and functions.

Interval arithmetic [7], can be used as the building block for the p.a.r. computations. However, our model does not have the requirement for accurate descriptions of the image intervals, and it sometimes proves more efficient to use more rough approximation schemes instead of the elaborate algorithms used in interval arithmetic.

5 Implementation

We will try to answer what advantages this model gives for implementations and if there are any visible shortcomings. Most of the discussion in this part is based on observations made during the creation of *RealLib*[1], an exact real number package based on the two-layer model.

We call this model a type-1 model, but when a user writes and executes a program that computes with reals, and ends up with one or several real numbers as results, their program will still be at least a type-2 program in order to successfully

[1] available at `http://www.brics.dk/~barnie/RealLib`

extract any property they need (instead of getting it if the current approximation allows). Where is the difference then?

With the classical ideas, implementations (such as *XR* [2], *ICReals* [4], *CORE* [14]) first run through the program to build expression trees, lambda terms or other descriptions of the computation, and start computing only after a request for a property is given, using these descriptions.

But isn't the description of the computation already present? That is, isn't the program code already enough description? Why do we need to build those *huge* expression trees or lambda terms when we already have a compact description of the solution in an elaborate language such as *C++* or *ML*? Moreover, why should we lose the information that the programmer or compiler is giving us about *common subexpressions* (leading to efficiency problems like the one in the introduction), *objects' lifetime and locality* (leading to higher memory requirements and need for garbage collection) and *preferred ordering* of the operations (leading to complications to both former points)?

In order to put this to practice, one has to be able to allow real functions defined on the type-1 level, and a means to do this is exactly what this model provides. Before that, an implementation of this idea was used in *iRRAM* [11], but the ideas in this work could not be placed in any theoretical framework. One of our model's objectives was to fit this system.

The problem with a type-1 implementation is that it is not very easy for the user to request specific properties of a real number which is the result of a computation, and the behaviour of the implementations regarding control flow make it very complicated to *display* or *present* the information gathered. *iRRAM*, which has previously proved to be the most efficient implementation of real number arithmetic, displays both the pros and cons of the type-1 approach. Our implementation, *RealLib*, aims to diminish the ill effects and preserve the advantages.

This is done by a hybrid approach in which the user's access to real numbers is implemented on two levels, one that works on partial approximations in order to *compute*, and one that deals with descriptions of expression terms as full type-1 information of real numbers and is meant to *extract* and *use* the results. The execution at the type-2 level is normal and programming for it is easy. There is no history maintenance at the partial approximation level and the computations are fast, especially in the lower precisions where history maintenance dominates the execution time in type-2 approaches. It is the programmer's choice which part of their application should run where; the system is offering redundant definitions of primitives on both levels to make this possible, and the translation from one to the other is easy, amounting simply to a change of the variable types in many cases.

[2] K. Briggs, `http://members.lycos.co.uk/keithmbriggs/XR.html`

More detailed information about our implementation can be found in [10].

6 Future work

A generalization of the ideas presented here to non-locally compact Polish spaces is to be derived. It will possibly provide insight to the still open problem of the right notion of feasibility in the general type-2 setting.

Another interesting question is how to allow intensionality in the type-1 model. The constructions we use in this paper to prove equivalences all break down in the case of intensionally-defined real functions. Figuring whether and how this can be accommodated would help to avoid holes in the domains of functions such as *atan2*, *sqrt* on complex numbers, etc.

The implementation has room for improvements, most importantly aiming at making the switch from machine-type precision to exact real numbers as painless as possible, i.e. to reduce the performance gap even more, well beyond the boundary that history maintenance sets for type-2-only systems.

References

[1] Cook, S., Urquart, A., Functional interpretations of feasibly constructive arithmetic. Ann. Pure Applied Logic **63**, pp. 103-200 (1993).

[2] Cook, Stephen A. Computability and complexity of higher type functions. Logic from computer science (Berkeley, CA, 1989), 51–72, Math. Sci. Res. Inst. Publ., **21**, Springer, New York, 1992.

[3] Edalat, Abbas; Sünderhauf, Philipp A domain-theoretic approach to computability on the real line. Theoret. Comput. Sci. **210** (1999), no. 1, 73–98.

[4] Edalat, A., Exact Real Number Computation Using Linear Fractional Transformations,
available at `http://www.doc.ic.ac.uk/~ae/exact-computation/`

[5] Howard, W.A., Hereditarily majorizable functionals of finite type. In: Troelstra (ed.), Metamathematical investigation of intuitionistic arithmetic and analysis, pp. 454-461. Springer LNM **344** (1973).

[6] Kapron, B. M.; Cook, S. A. A new characterization of type-2 feasibility. SIAM J. Comput. **25** (1996), no. 1, 117–132.

[7] Kearfott, Baker R. Interval computations: introduction, uses, and resources. Euromath Bull. **2** (1996), no. 1, 95–112.

[8] Ko, K.-I., Complexity theory of real functions. Birkhäuser, Boston-Basel-Berlin, x+309 pp., 1991.

[9] Kohlenbach, U. Lecture Notes: Proof Interpretations and the Computational Content of Proofs (Draft, May 2003, ii+165pp.)
available at `http://www.brics.dk/~kohlenb/`

[10] Lambov, B., A two-layer approach to the computability and complexity of real functions. Computability and complexity in analysis (Cincinnati, 2003), 279–302, Informatik Berichte, **302** (8/2003), Fernuniversität Hagen, 2003
see also `http://www.brics.dk/~barnie/`

[11] Müller, N., The iRRAM: Exact arithmetic in C++. in Computability and complexity in analysis. (Swansea, 2000). Lecture Notes in Computer Science, **2064**. Springer-Verlag, Berlin, 2001. viii+395 pp.
see also `http://www.informatik.uni-trier.de/iRRAM/`

[12] Skordev, D., Characterization of the computable real numbers by means of primitive recursive functions. Computability and complexity in analysis (Swansea, 2000), 296–309, Lecture Notes in Computer Science, **2064**, Springer-Verlag, Berlin, 2001. viii+395 pp.

[13] Weihrauch, K., Computable Analysis. Springer, Berlin 2000.

[14] Yap, Chee, Towards Exact Geometric Computation. Computational Geometry : Theory and application, **3-23**, Sep 1997.
see also `http://www.cs.nyu.edu/exact/core/`

Recent BRICS Report Series Publications

RS-03-50 Branimir Lambov. *A Two-Layer Approach to the Computability and Complexity of Real Numbers*. December 2003. 16 pp.

RS-03-49 Marius Mikucionis, Kim G. Larsen, and Brian Nielsen. *Online On-the-Fly Testing of Real-time Systems*. December 2003. 14 pp.

RS-03-48 Kim G. Larsen, Ulrik Larsen, Brian Nielsen, Arne Skou, and Andrzej Wasowski. *Danfoss EKC Trial Project Deliverables*. December 2003. 53 pp.

RS-03-47 Hans Hüttel and Jiří Srba. *Recursive Ping-Pong Protocols*. December 2003. To appear in the proceedings of 2004 IFIP WG 1.7, ACM SIGPLAN and GI FoMSESS Workshop on Issues in the Theory of Security (WITS'04).

RS-03-46 Philipp Gerhardy. *The Role of Quantifier Alternations in Cut Elimination*. December 2003. 10 pp. Extends paper appearing in Baaz and Makowsky, editors, *European Association for Computer Science Logic: 17th International Workshop*, CSL '03 Proceedings, LNCS 2803, 2003, pages 212-225.

RS-03-45 Peter Bro Miltersen, Jaikumar Radhakrishnan, and Ingo Wegener. *On converting CNF to DNF*. December 2003. 11 pp. A preliminary version appeared in Rovan and Vojtás, editors, *Mathematical Foundations of Computer Science: 28th International Symposium*, MFCS '03 Proceedings, LNCS 2747, 2003, pages 612–621.

RS-03-44 Anna Gál and Peter Bro Miltersen. *The Cell Probe Complexity of Succinct Data Structures*. December 2003. 17 pp. An early version of this paper appeared in Baeten, Lenstra, Parrow and Woeginger, editors, *30th International Colloquium on Automata, Languages, and Programming*, ICALP '03 Proceedings, LNCS 2719, 2003, pages 332–344.

RS-03-43 Mikkel Nygaard and Glynn Winskel. *Domain Theory for Concurrency*. December 2003. 45 pp. To appear in a *Theoretical Computer Science* special issue on Domain Theory.