Extracting efficient exact real number computation from proofs in constructive type theory

MICHAL KONEČNÝ, *Aston University, College of Engineering and Physical Sciences, Birmingham B4 7ET, UK.*
E-mail: m.konecny@aston.ac.uk

SEWON PARK, *Kyoto University, Graduate School of Informatics, Kyoto 606-8501, Japan.*
E-mail: sewon@fos.kuis.kyoto-u.ac.jp

HOLGER THIES, *Kyoto University, Graduate School of Human and Environmental Studies, Kyoto 606-8501, Japan.*
E-mail: thies.holger.5c@kyoto-u.ac.jp

Abstract

Exact real computation is an alternative to floating-point arithmetic where operations on real numbers are performed exactly, without the introduction of rounding errors. When proving the correctness of an implementation, one can focus solely on the mathematical properties of the problem without thinking about the subtleties of representing real numbers. We propose a new axiomatization of the real numbers in a dependent type theory with the goal of extracting certified exact real computation programs from constructive proofs. Our formalization differs from similar approaches, in that we formalize the reals in a conceptually similar way as some mature implementations of exact real computation. Primitive operations on reals can be extracted directly to the corresponding operations in such an implementation, producing more efficient programs. We particularly focus on the formalization of partial and nondeterministic computation, which is essential in exact real computation. We prove the soundness of our formalization with regards to the standard realizability interpretation from computable analysis and show how to relate our theory to a classical formalization of the reals. We demonstrate the feasibility of our theory by implementing it in the Coq proof assistant and present several natural examples. From the examples we have automatically extracted Haskell programs that use the exact real computation framework AERN for efficiently performing exact operations on real numbers. In experiments, the extracted programs behave similarly to native implementations in AERN in terms of running time and memory usage.

Keywords: constructive real numbers, formal proofs, exact real number computation, program extraction

1 Introduction

Exact real computation is an elegant programming paradigm in which real numbers and other continuous mathematical structures are treated as basic entities that can be manipulated exactly. This is usually realized by adding a datatype for reals and arithmetic operations on them as primitives in programming languages. All computations on real numbers are done exactly, without introducing rounding errors, and the implementation usually allows to output finite approximations of real numbers up to any desired absolute precision. Although the approach is less efficient than optimized hardware-based floating-point calculations, implementations in exact real computation are by design more reliable than the former which is desirable in safety-critical application domains where correctness is more important than efficiency. Furthermore, comparably efficient implementations

Vol. 00, No. 00, © The Author(s) 2024. Published by Oxford University Press. All rights reserved. For permissions, please e-mail: journals.permission@oup.com
https://doi.org/10.1093/logcom/exae066

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

exist that try to minimize the additional overhead [1, 26, 38]. These implementations are successfully used to approximate numerical values up to very high precision in practice.

Proof assistants and automatic verification methods are increasingly used in computer science to verify the correctness of hardware and software [46]. Numerical computations, often at the intersection between software and the physical world, are in many cases safety-critical and thus their verification is a very active area of research [7, 8, 11, 19, 34]. Verifying floating point programs is, however, challenging as one not only has to make sure that the mathematical statements are correct, but also needs to deal with issues of the approximate implementation of real numbers [10, 36]. In exact real computation, on the other hand, the implementation behaves closely to the mathematical theory, and one can therefore focus on the purely mathematical statements, facilitating the process of formal verification.

When it comes to formally verifying programs, one can broadly distinguish between two main approaches [16]. The first one is classical program verification, that is, to first write the program and then, in a second step, formally verify that it meets some specification. The second one is to extract programs directly from constructive proofs, that is, to integrate programming, specification and verification in a single step. Arguably, the second approach is more elegant and, due to peculiarities of computing with real numbers, might even be more natural when dealing with exact real computation. Still, giving up on writing programs directly, some control over the exact form of the algorithm is lost and thus fine-grained optimization becomes more difficult. Results regarding extraction of exact real computation programs are therefore mostly theoretical and do not produce code that is efficient enough to be used in practice [44]. Moreover, subtleties like partiality, concurrency and nondeterministic computation that arise naturally and are unavoidable in exact real computation have been less studied in the context of program extraction [3]. In this work, we suggest a new formalization to overcome some of the above mentioned problems of program extraction. Instead of formalizing a concrete implementation of constructive real numbers, we suggest a high-level axiomatization of a constructive real number type, formalizing real numbers conceptually similarly to how they are treated in some of the more efficient implementations of exact real computation. The goal is to extract programs where the constructive real type, and its axiomatically defined basic operations such as arithmetic or limits, are directly mapped to corresponding types and operations in an exact real computation framework. That is, instead of re-implementing and verifying basic real number operations, we trust the implementation of a core of simple real number operations and to verify programs using those operations under the assumption that they are correctly implemented. Concretely, utilising this mapping and program extraction techniques, we obtain certified programs over an implementation of exact real computation from correctness proofs. The approach also provides a certain amount of independence of the concrete implementation of real numbers and thus allows to easily switch the underlying framework.

1.1 Overview and main contributions

In this paper, we define an axiomatization of the reals in a constructive dependent type theory that allows program extraction. Our axioms model real numbers in a conceptually similar way as mature implementations of exact real computation. We further allow a certain amount of classical logic in proofs that do not have computational content. As we introduce several axioms in our theory, an important question is how we make sure that this kind of formalization is sound. We argue that our formalization is sound by extending a realizability interpretation in the category of assemblies over Kleene's second algebra [5, 21]. The theoretical framework for this kind of reasoning is given by computable analysis and the theory of representations [31, 54]. We describe the necessary background in Section 2.

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

As computable operations are necessarily continuous, nondeterminism is central in exact real computation. We model nondeterministic computation as a monad that we introduce in Section 3. For any type $X : \mathsf{Type}$, $\mathsf{M}X : \mathsf{Type}$ stands for the type of nondeterministic computations in X. The idea is that when proving a theorem of the form $\Pi(x : \mathsf{R}).\ Px \to \mathsf{M}\Sigma(y : \mathsf{R}).\ Qxy$, a user automatically gets an exact real computation program that for any input $x \in \mathbb{R}$ such that $P(x)$ holds, nondeterministically computes $y \in \mathbb{R}$ such that $Q(x, y)$ holds.

In Section 4, we define the real number axiomatization and some simple extensions to complex numbers and general Euclidean spaces. One of the defining features of exact real computation, making it more powerful than e.g. symbolic or algebraic computation, is the ability to construct real numbers by limits of certain user-defined sequences [13, 39]. Axiomatizing the limit itself is not difficult but the situation becomes more involved for the case when nondeterminism is present in the computation, and how to define such a nondeterministic limit operation has been under debate. In the current work, we specify *nondeterministic dependent choice*, a simple, natural principle of the nondeterminism itself that makes the limit operation suggested in [44] and other forms of nondeterministic limits derivable. In Section 5, we prove the soundness of our axioms by extending the realizability interpretation from Section 2.2.

As there are already formulations of classical real numbers in proof assistants, from a practical point of view it would be desirable to transfer some of the proofs done over a classical axiomatization to our setting. We address this in Section 6 by defining a $\mathsf{relator}$ operation that lets us relate classical and constructive results in our type theory.

We present several larger examples from computable analysis in Section 7. Furthermore, we implemented all of the theory (apart from the soundness in Section 5, which we state as a metatheorem) in the Coq proof assistant. We describe the details of the implementation in Section 8. The implementation allows to extract Haskell programs from our proofs using Coq's code extraction mechanism. In the extracted programs, primitive operations on the reals are mapped to operations in the exact real computation framework AERN [26], which is written and maintained by one of the authors. We extracted code for several of the examples and measured its performance. We report that the extracted programs perform efficiently, having only a small overhead compared to hand-written implementations.

1.2 Related work

Implementations of constructive real numbers exist in many modern proof assistants (see e.g. [9] for an overview). In Coq, in particular, there is a substantial formalization of constructive reals [18] and a vast amount of subsequent work on exact real computation (e.g. [30, 40–42]). In newer versions, the Coq standard library also provides an abstract interface `ConstructiveReals`, which is based on some of that work. This approach of defining a type of constructive real numbers, and declaring its abstract interface, differs from our approach of axiomatizing constructive real numbers. With the absence of quotient types in Coq, using setoids becomes necessary to define a constructive real number type. There, to refer to the ordinary equality of real numbers x, y, a user-defined relation $x \sim y$ is used whereas the built-in identity $x = y$ refers to the intentional equality of real numbers e.g. the Cauchy sequences coinciding. Our approach can be seen as axiomatizing the quotient over $\sim$ making the equality of real numbers become the usual equality notion of real numbers. As a result, the intuitive meaning of the real number type coincides with the classical set of real numbers. One side-effect of identifying the extensional and intensional equalities of real numbers is that nondeterminism becomes essential. Still, we think that this resembles how real number computations are usually written and reasoned about; see [43] and [14].

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

Apart from Coq and other type-theory based frameworks, extraction of exact real computation programs has been studied in many other settings. For example, [4, 35] deal with extracting exact real computation programs in logic-based systems. These systems have some advantages when it comes to coinductive definitions, which is known to pose some difficulties in Coq, but has useful applications in exact real computation [2]. However, the infrastructure for these systems is less developed than for larger systems, making it difficult to prove more advanced results and to extract efficient programs. Another difference to our work is that these systems work with a concrete representation of the real numbers and extract programs over this representation, which is exactly what we want to avoid with our axiomatic approach.

Recently, there also has been a growing interest in formal verification in the computable analysis community. Most of these works follow the program verification approach instead of extracting programs directly from proofs. For example, [43] formalize a simple imperative programming language for exact real computation as a framework for program verification and [51] present Incone another Coq library on computable analysis. Note that Incone has a different scope than the library presented in the current paper: it formalizes some of the theory of classical computable analysis, for example, the standard realizability notion and is used mostly to formalize more theoretical results. Nonetheless, Incone also allows to define exact real computation algorithms by specifying them directly over the respective representations.

This paper is an extended version of the conference paper [27] that appeared in the proceedings of the 27th Workshop on Logic, Language, Information and Computation. In the meantime, our theoretical framework has evolved significantly. Most importantly, we suggest a different set of axioms for the nondeterminism monad that is expressive enough to define various notions of nondeterministic completeness useful in applications. Some details about this extension have been presented at the 14th International Symposium on NASA Formal Methods and published in the proceedings [28]. To offer a comprehensive overview over the current state of our formalization, we decided to include the new results and update our presentation accordingly.

Although there is some significant overlap with the conference versions, we have incorporated all details into the current paper to ensure its self-contained nature. Let us briefly give an overview on how the current paper relates to and extends the conference proceedings. First, we provide a much more detailed background section. We think that this allows interested readers without background in computable analysis or type theory to follow our work. The basic axiomatization of the Kleeneans (Section 3.1) and the real numbers (Section 4) has been described in [27, Section 3]. We extend the presentation to include the complete axiomatization and add additional details and proofs. The Soundness Theorem (metaTheorem 1) has only been sketched briefly in the previous versions (cf. [27, Lemma 1]). Here, we dedicate Section 5 to prove the soundness of our axioms in detail, including the new axioms regarding nondeterminism from [28]. We also include some details about how we formalize complex numbers and general Euclidean spaces (Section 4.5) which has been omitted from the previous works. The relator operation from Section 6, used to relate proofs over classical formalizations of real numbers to our new type, has been briefly described in [27, Section 4]. We reformulated this section to make the presentation more clear and complete. We also added an additional example (example 4) and a detailed proof of example 5. The basic axiomatization of the nondeterminism monad (Section 3.2) can be found in [28, Section 3]. We kept most of this formalization in the present work and put it at the beginning of the section (Section 3.2, Section 3.3), only slightly improving the presentation. However, after that we provide some new results that have not been discussed in any of the previous works. Most importantly, we added a new Section about nondetermism logic (Section 3.4), which demonstrates further applications of our theory and is useful to make the later presentation more concise. We also added two new examples on the use of the nondeterminism monad and lemma 1, which shows how our new formalization relates to the

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

one presented in [27, Section 3.1]. Combining the limit operation with the nondeterminism monad as described in Section 4.3, Section 4.4 has also been presented in [28]. Here, we added proofs for the main results (Theorems 1 and 2). We also added a new Examples section, extending what has only have been briefly described in [27] as part of the implementation section. To this end, we reformulated all the examples in our type-theoretical framework, added more details for the example of the fast square root algorithm (Section 7.3) and combined the section with the examples from [28, Section 5.1 and 5.2] that demonstrate the use of nondeterministic limits. Finally, the implementation section (Section 8) has been vastly extended, containing a concise description of our Coq implementation.

2 Background

Before describing our axiomatization let us briefly summarize the necessary background from computable analysis and define the type-theoretical setting in which we present our results.

2.1 Computable analysis

To compute over uncountable mathematical structures such as real numbers exactly, computable analysis takes *assemblies* over Kleene's second algebra (assemblies for short in this paper) as the basic data type [5, 21, 53].[1] An assembly is a pair of a set X and a relation $\Vdash \subseteq \mathbb{N}^{\mathbb{N}} \times X$, which is surjective in that $\forall x \in X.\ \exists \varphi.\ \varphi \Vdash x$ holds, in infix notation. We call $\varphi \in \mathbb{N}^{\mathbb{N}}$ a realizer of an abstract entity $x \in X$ if $\varphi \Vdash x$ holds. We often write assemblies in boldface capitalized letters $\mathbf{X}, \mathbf{Y}, \cdots$. For an assembly $\mathbf{X}$, we write $|\mathbf{X}|$ for the underlying set of $\mathbf{X}$ and $\Vdash_{\mathbf{X}}$ for the realization relation of $\mathbf{X}$. We often write $x \in \mathbf{X}$ for $x \in |\mathbf{X}|$ to simplify the presentation.

Given two assemblies $\mathbf{X}$ and $\mathbf{Y}$, a function $f : |\mathbf{X}| \to |\mathbf{Y}|$ is said to be computable if there is a computable (in the sense of type-2 computability [54]) partial function $\tau :\subseteq \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ that tracks f, i.e. for any $x \in \mathbf{X}$ and its realizer φ, $\tau(\varphi)$ is a realizer of $f(x)$. Collecting assemblies as objects and computable functions between them as morphisms, the category of assemblies $\mathsf{Asm}(\mathcal{K}_2)$ forms a quasitopos.

An example of an assembly is the Cauchy assembly $\mathbf{R}_{\text{Cauchy}}$ over the reals where $\varphi \in \mathbb{N}^{\mathbb{N}}$ gets to be a realizer of $x \in \mathbb{R}$ if and only if φ encodes a sequence of rationals converging rapidly towards x. That is, when we specify $\eta_{\mathbb{Q}} : \mathbb{N} \to \mathbb{Q}$ to be the underlying encoding of rationals, it holds that

$$|x - \eta_{\mathbb{Q}}(\varphi(n))| \leq 2^{-n} \iff \varphi \Vdash_{\mathbf{R}_{\text{Cauchy}}} x.$$

Under the realization relation, the field arithmetic of reals is computable. That means the arithmetic operations get computed exactly without any rounding errors by the definition of the realization relation.

An inevitable side-effect of this approach is partiality. Consider comparing the order of two real numbers from the Cauchy assembly. The naive procedure of iterating through the entries of the realizers of the two real numbers will fail to terminate when the two real numbers are identical. In fact, whichever realizability relation of reals we take, comparisons of real numbers are only partially computable [54, Theorem 4.1.16].

A partial function $f :\subseteq X \to Y$ is called computable with regards to the realization relations $\Vdash_X$ and $\Vdash_Y$ at its restricted domain $S \subseteq X$ if it, as a total function $f : S \to Y$, is computable with regards

[1] Assemblies are generalizations of represented sets [31, 54], which are exactly the assemblies where the surjective relations are required to be partial surjective functions. The terminology *multi-representation* [47] may be more familiar to some readers.

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

$\hat{\wedge}$	tt	ff	$\bot$
tt	tt	ff	$\bot$
ff	ff	ff	ff
$\bot$	$\bot$	ff	$\bot$

$\hat{\vee}$	tt	ff	$\bot$
tt	tt	tt	tt
ff	tt	ff	$\bot$
$\bot$	tt	$\bot$	$\bot$

$\hat{\neg}$	tt	ff	$\bot$
	ff	tt	$\bot$

FIGURE 1. The Kleene three-valued logic operations, where $\bot$ represents undefinedness.

to $\Vdash_{S\subseteq X}$ and $\Vdash_Y$. The realization relation $\Vdash_{S\subseteq X}$ on S is a relation induced by the subset embedding from $\Vdash_X$ by

$$\varphi \Vdash_{S\subseteq X} x :\Leftrightarrow \varphi \Vdash_X x$$

With regards to the Cauchy assembly, the order comparison $<$ as a binary function to the Booleans is only partially computable when it is not defined at $\Delta_{\mathbb{R}} := \{(x,x) \mid x \in \mathbb{R}\}$.

Classically, partial functions $f :\subseteq X \to Y$ are often handled as a total function to a lifted codomain $f' : X \to Y \cup \{\bot\}$ where the new codomain contains an extra value $\bot$ which denotes that the function received a nonvalid input. To be precise, let

$$f' := x \mapsto \begin{cases} f(x) & \text{if } x \in S \\ \bot & \text{otherwise,} \end{cases}$$

where $S \subseteq X$ is the actual domain of the partial function $f :\subseteq X \to Y$. However, in order to do the same in our setting, to make the lifted codomain an assembly, we need to consider the computational behaviour of $\bot$ explicitly as we need to define the realizers of $\bot$. It means, up to a way of defining the realizers of partialities, in this setting, partial functions can be classified by their computational behaviour when they accept nonspecified inputs [32, Ch. 4].

Consider, for example, the problem of comparing the order of real numbers. Regarding it as a computable partial function that is not defined at $\Delta_{\mathbb{R}}$, following the naive procedure of distinguishing the real numbers from their realizers, we can say that the computational behaviour of the function when it receives inputs from $\Delta_{\mathbb{R}}$ is diverging, e.g. nontermination. Let *Kleenean* **K** be the assembly of $\{ff, tt, \bot\}$ where an infinite sequence of zeros realizes $\bot$, an infinite sequence that starts with 1 after a finite prefix of zeros realizes ff, and an infinite sequence that starts with 2 after a finite prefix of zeros realizes tt (see e.g. [25, Example 3]). The assembly **K** can be seen as a generalization of the Booleans by adding an explicit state of divergence $\bot$. Comparison in the Cauchy assembly is computable as a function to **K**:

$$x < y := \begin{cases} tt & \text{if } x < y, \\ ff & \text{if } y < x, \\ \bot & \text{otherwise.} \end{cases}$$

That means, **K** can be used to classify partial Boolean functions whose specification at nonspecified inputs is to diverge (nontermination), including the order comparison of Cauchy reals. As the name suggests, the assembly **K** makes the Kleene three-valued logic operations, shown in Fig. 1, computable.

An assembly of reals satisfying the below computability conditions is called *effective*.

FACT 1. For real numbers, there is a unique assembly up to isomorphism that satisfies the computability-theoretic structure of real numbers [20]:

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

1. The constants $0, 1 \in \mathbb{R}$ are computable.
2. The arithmetical operations $+, -, \times$ are computable.
3. The multiplicative inversion / is computable as a partial function that is not defined at 0.
4. The order relation $<$ as a partial function that is not defined at $\{(x,x) \mid x \in \mathbb{R}\}$ is computable. However, its lifted version in $\mathbf{K}$ is computable.
5. The limit operation defined at rapidly converging sequences is computable.

A concrete example of such an assembly of reals is, of course, the Cauchy reals. Hence, any effective axiomatization of real numbers has to capture this computational structure.

As the usual comparisons are partial, nondeterminism becomes essential in exact real computation [33]. For two assemblies of X and Y, a multivalued function $f : X \rightrightarrows Y$, which is basically a nonempty set-valued function, is computable if there is a computable function that takes a realizer φ of $x \in X$ and computes a realizer of any $y \in f(x)$. An example is the multivalued soft comparison [12]:

$$x <_k y = \{tt \mid x < y + 2^k\} \cup \{ff \mid y < x + 2^k\}.$$

The above total multivalued function approximates the order relation. It is tracked by evaluating two partial comparisons $x < y + 2^k$ and $y < x + 2^k$ in parallel, returning tt if $x < y + 2^k = tt$, and ff if $y < x + 2^k = tt$. It is nondeterministic in the sense that for the same x and y, but with different realizers, which of the tests terminates first may vary. Exact real number computation software such as [26, 38] further offer operators like $\mathbf{select} :\subseteq \mathbf{K} \times \mathbf{K} \rightrightarrows \mathbf{2}$ such that $\mathbf{select}(k_1, k_2) \ni tt$ iff $k_1 = tt$ and $\mathbf{select}(k_1, k_2) \ni ff$ iff $k_2 = tt$ as a primitive operation for generating nondeterminism. Here, $\mathbf{2}$ is the canonical Boolean assembly.

2.2 Type theory and realizability

Type theory is a foundation of mathematics consisting of rules for forming new types and constructing new terms of the types. In this paper, we work with a simple extensional dependent type theory that admits categorical semantics, i.e. a realizability interpretation in $\mathsf{Asm}(\mathcal{K}_2)$ [22, 45, 52]. That is, each closed type X is mapped to an assembly $[\![X]\!]$ and each closed term $x : X$ is mapped to a computable point (function) $[\![x]\!] : \mathbf{1} \to [\![X]\!]$. Hence, the type theory can be used as a logical language of computable analysis.

We work with a type theory that provides the basic types: $\mathsf{0}, \mathsf{1}, \mathsf{2}, \mathsf{N}, \mathsf{Z}$, an à la Russel universe of classical propositions Prop, and an à la Russel universe of types Type (we omit the technical details necessary to avoid the usual paradoxes by e.g. imposing a hierarchy of types). We interpret $\mathsf{0}$ as the initial assembly $\mathbf{0}$, the assembly of the empty set, $\mathsf{1}$ as a final assembly, the trivial assembly of a singleton set $\mathbf{1}$, $\mathsf{2}$ as the canonical assembly over $\{tt, ff\}$, N as the canonical assembly over $\mathbb{N}$, and Z as the canonical assembly over $\mathbb{Z}$. Here, triviality means that the underlying realization relation is the entire set. We also assume sufficient enough operators over the basic types and interpret them appropriately to the computable functions in $\mathsf{Asm}(\mathcal{K}_2)$ including primitive recursors over the finite and countable types.

The interpretations of type formations are done in a standard way. For two types X and Y, the (co)product type $(X + Y)$ $X \times Y$ is interpreted as the (co)product of the interpretations of X and Y. The function type $X \to Y$ is interpreted as the exponentiation of the interpretation of X over the interpretation of Y. Given a family of assemblies $\mathcal{F} : |\mathbf{X}| \to \mathsf{Asm}(\mathcal{K}_2)$ indexed by $|\mathbf{X}|$, consider the two assemblies: $\pi(\mathcal{F})$ of the set of dependent functions $\{f : |\mathbf{X}| \to \bigcup_x |\mathcal{F}(x)| \mid \forall x.\, f(x) \in |\mathcal{F}(x)|\}$ and $\sigma(\mathcal{F})$ of the set of dependent pairs $\{(x,y) \mid x \in |\mathbf{X}| \wedge y \in |\mathcal{F}(x)|\}$. The realization relations are defined by $\varphi \Vdash_{\pi(\mathcal{F})} f :\Leftrightarrow \forall \psi \Vdash_{\mathbf{X}} x.\ \eta_\varphi(\psi) \Vdash_{\mathcal{F}(x)} f(x)$ and $\langle \varphi_1, \varphi_2 \rangle \Vdash_{\sigma(\mathcal{F})} (x,y) :\Leftrightarrow \varphi_1 \Vdash_{\mathbf{X}} x \wedge$

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

$\varphi_2 \Vdash_{\mathcal{F}(x)} y$. Here, η_ϕ denotes the underlying universal type-2 Turing machine indexed by $\phi \in \mathbb{N}^\mathbb{N}$ and $\langle \varphi_1, \varphi_2 \rangle \in \mathbb{N}^\mathbb{N}$ denotes the Cantor paring of $\varphi_1, \varphi_2 \in \mathbb{N}^\mathbb{N}$. More details of the type-2 computability theory can be found at [54].

When we have a type family $P : X \to \mathsf{Type}$, we interpret its Π-type $\Pi(x : X).\, P\,x$ as the π-assembly of the family of assemblies expressed by P, and for a type family $Q : X \to \mathsf{Type}$, $\Sigma(x : X).\, Q\,x$ is interpreted as the σ-assembly of the family of assemblies expressed by Q.

We regard Prop as a universe of classical propositions. We interpret Prop as the trivial assembly of $\{\mathbf{0}, \mathbf{1}\}$ where $\mathbf{0}$ corresponds to $\mathsf{False} : \mathsf{Prop}$ and $\mathbf{1}$ corresponds to $\mathsf{True} : \mathsf{Prop}$. We further suppose that Prop is a type universe that is closed under $\to, \times, \Pi$. We often emphasize the property of Prop that it is a universe of (classical) propositions, by writing $\Rightarrow, \wedge, \forall$ instead of $\to, \times, \Pi$. However, note that Prop is not closed under $+$ and Σ. We assume the Prop-level counterpart of the constructions $\vee$ and $\exists$ where $\neg\neg(X + Y) \simeq X \vee Y$ and $\neg\neg\Sigma(x : X).\, Y \simeq \exists(x : X).\, Y$ get provable. Here, $\simeq$ denotes type-theoretic equivalent which in our case is simply category-theoretic isomorphism.

In order to make Prop classical, we assume the (classical) law of excluded middle

$$\Pi(P : \mathsf{Prop}).\, P \vee \neg P \qquad \text{(Axiom TT1)}$$

(where $\neg P :\equiv P \to \mathsf{False}$), the (classical) propositional extensionality

$$\Pi(P, Q : \mathsf{Prop}).\, (P \leftrightarrow Q) \to P = Q \qquad \text{(Axiom TT2)}$$

(where $P \leftrightarrow Q :\equiv (P \to Q) \times (Q \to P)$), and the (classical) countable choice of the form

$$\Pi(X : \mathsf{Type}).\, \Pi(P : \mathsf{N} \to X \to \mathsf{Prop}).\, (\Pi(n : \mathsf{N}).\, \exists(x : X).\, P\,n\,x) \to \exists(f : \mathsf{N} \to X).\, \Pi(n : \mathsf{N}).\, P\,n\,(f\,n). \qquad \text{(Axiom TT3)}$$

Here, and on all later occasions when we introduce axioms, we sometimes just write the type of the axiom instead of the full axiom, by which we mean that the type is inhabited. Note that the axioms are validated under the interpretation in $\mathsf{Asm}(\mathcal{K}_2)$.

We assume that the identity types $=$ are in Prop where we have extensionality, that we interpret $x = y$ as $\mathbf{0}$ when the interpretations of x and y are not identical and as $\mathbf{1}$ when the interpretations are identical.

We also assume the general functional extensionality

$$\Pi(X : \mathsf{Type}).\, \Pi(P : X \to \mathsf{Type}).\, \Pi(f, g : \Pi(x : X).\, P\,x).\, \big(\Pi(x : X).\, f\,x = g\,x\big) \to f = g \qquad \text{(Axiom TT4)}$$

and the Markov principle

$$\Pi(f : \mathsf{N} \to \mathsf{Prop}).\, \big(\Pi(n : \mathsf{N}).\, (f\,n) + \neg(f\,n)\big) \to (\exists(n : \mathsf{N}).\, f\,n) \to \Sigma(n : \mathsf{N}).\, f\,n. \qquad \text{(Axiom TT5)}$$

3 Nondeterminism

We propose an axiomatization of the nondeterminism considered in computable analysis as a monad $\mathsf{M} : \mathsf{Type} \to \mathsf{Type}$ in our type theory such that $\mathsf{M}X : \mathsf{Type}$ stands for the type of nondeterministic computations in $X : \mathsf{Type}$. The monad will be used intensively as, for example, $f : A \to \mathsf{M}B$ to denote a computable multivalued function $A \rightrightarrows B$, $\mathsf{M}(A + B)$ to denote a nondeterministic decision

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

procedure deciding either A or B, and $\mathsf{M}\Sigma(x : A).\ B$ to denote a nondeterminstic existence of $x : A$ such that $B(x)$.

The aim is to achieve the interpretation that when we have a closed term of type $\Pi(x : A).\ P(x) \to \mathsf{M}\Sigma(y : B).\ Q\,x\,y$, it automatically gives us a nondeterministic program that computes the partial multivalued function

$$f :\subseteq [\![A]\!] \rightrightarrows [\![B]\!]$$

which is defined on $\{x \mid [\![P]\!](x)\}$ satisfying $\forall y \in f(x).\ [\![Q]\!](x, y)$.

The goal of this section is to propose a set of axioms that equips the type theory with the monad. As in computable analysis basic nondeterminism is often introduced through Kleeneans, we first axiomatize the assembly K in our type theory. Then, believing that the type theory must be rich enough to do classical reasoning on the nondeterministic values, we characterize the nondeterminism monad as a sub-object of the classical nonempty power-set monad, which can be constructed within the type theory. The latter part of the interpretation is dealt with in Section 5.

3.1 Axiomatizing Kleeneans

To axiomatize the Kleenans as described in Section 2, we assume that there is a type $\mathsf{K} : \mathsf{Type}$ of Kleeneans and that there are two *distinct* elements $\mathsf{true} : \mathsf{K}$ and $\mathsf{false} : \mathsf{K}$. That is, we introduce the following four axioms:

$$\mathsf{K} : \mathsf{Type} \tag{Axiom K1}$$

$$\mathsf{true} : \mathsf{K} \tag{Axiom K2}$$

$$\mathsf{false} : \mathsf{K} \tag{Axiom K3}$$

$$\mathsf{true} \neq \mathsf{false} \tag{Axiom K4}$$

Let us define the abbreviations $\lceil t \rceil : \mathsf{Prop} :\equiv t = \mathsf{true}$ and $\lfloor t \rfloor : \mathsf{Prop} :\equiv t = \mathsf{false}$. For a Kleenean $t : \mathsf{K}$, we further write $t \downarrow$ if t is defined, i.e. for $\lceil t \rceil \vee \lfloor t \rfloor : \mathsf{Prop}$. This allows us to introduce the standard logical operators (negation, conjunction and disjunction) on Kleeneans, which follows the three-valued logic:

$$\hat{\neg} : \mathsf{K} \to \mathsf{K} \tag{Axiom K5}$$

$$\hat{\vee} : \mathsf{K} \to \mathsf{K} \to \mathsf{K} \tag{Axiom K6}$$

$$\hat{\wedge} : \mathsf{K} \to \mathsf{K} \to \mathsf{K} \tag{Axiom K7}$$

$$\lceil \hat{\neg} x \rceil = \lfloor x \rfloor \ \text{ and } \lfloor \hat{\neg} x \rfloor = \lceil x \rceil \tag{Axiom K8}$$

$$\lceil x \mathbin{\hat{\wedge}} y \rceil = (\lceil x \rceil \wedge \lceil y \rceil) \ \text{ and } \lfloor x \mathbin{\hat{\wedge}} y \rfloor = (\lfloor x \rfloor \vee \lfloor y \rfloor) \tag{Axiom K9}$$

$$\lceil x \mathbin{\hat{\vee}} y \rceil = (\lceil x \rceil \vee \lceil y \rceil) \ \text{ and } \lfloor x \mathbin{\hat{\vee}} y \rfloor = (\lfloor x \rfloor \wedge \lfloor y \rfloor) \tag{Axiom K10}$$

If a Kleenean is *defined*, we assume that we can do branching on its value.

$$\Pi(x : \mathsf{K}).\ x\downarrow \to \lceil x \rceil + \lfloor x \rfloor. \tag{Axiom K11}$$

In many cases, we do not work directly with Kleeneans. Instead, we call a proposition $P : \mathsf{Prop}$ semi-decidable if there is a Kleenean t that identifies P:

$$\mathsf{semiDec}(P) :\equiv \Sigma(t : \mathsf{K}).\ P = \lceil t \rceil$$

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

For example, in Section 4 we introduce the comparison operator $<: \mathsf{R} \to \mathsf{R} \to \mathsf{Prop}$ over the reals and assume $\Pi(x, y : \mathsf{R}).\ \Sigma(t : \mathsf{K}).\ \lceil t \rceil = (x < y)$ to say that the operator is semi-decidable.

3.2 Basic axiomatization of the nondeterminism monad

Nondeterminism is expressed by a monad in our type theory such that when we have a type $X : \mathsf{Type}$, we automatically have its nondeterministic version $\mathsf{M}X : \mathsf{Type}$.

We assume that there is a type constructor $\mathsf{M} : \mathsf{Type} \to \mathsf{Type}$, a function lift $\mathsf{lift}^{\mathsf{M}}$, a unit $\mathsf{unit}^{\mathsf{M}}$ and a multiplication $\mathsf{mult}^{\mathsf{M}}$.

$$\mathsf{M} : \mathsf{Type} \to \mathsf{Type} \tag{Axiom M1}$$

$$\mathsf{unit}^{\mathsf{M}} : \Pi(X : \mathsf{Type}).\ X \to \mathsf{M}\,X \tag{Axiom M2}$$

$$\mathsf{mult}^{\mathsf{M}} : \Pi(X : \mathsf{Type}).\ \mathsf{M}\,(\mathsf{M}\,X) \to \mathsf{M}\,X \tag{Axiom M3}$$

$$\mathsf{lift}^{\mathsf{M}} : \Pi(X, Y : \mathsf{Type}).\ (X \to Y) \to (\mathsf{M}\,X \to \mathsf{M}\,Y) \tag{Axiom M4}$$

To simplify the presentation, let us write the type arguments to the unit, lift, and multiplication in subscript. For example, we write $\mathsf{unit}^{\mathsf{M}}_X$ instead of $\mathsf{unit}^{\mathsf{M}}X$.

We add the following axioms to ensure that the type constructor, function lift, unit and multiplication indeed form a monad. Namely, $\mathsf{unit}^{\mathsf{M}}$ and $\mathsf{mult}^{\mathsf{M}}$ are natural transformations and that the monad coherence conditions hold, i.e.

$$\mathsf{lift}^{\mathsf{M}}_{X,Y} f\ (\mathsf{unit}^{\mathsf{M}}_X\ x) = \mathsf{unit}^{\mathsf{M}}_Y\ (f\ x) \tag{Axiom M5}$$

$$\mathsf{mult}^{\mathsf{M}}_Y\ (\mathsf{lift}^{\mathsf{M}}_{\mathsf{M}X,\mathsf{M}Y}\ (\mathsf{lift}^{\mathsf{M}}_{X,Y} f)\ y) = (\mathsf{lift}^{\mathsf{M}}_{X,Y} f)\ (\mathsf{mult}^{\mathsf{M}}_X y) \tag{Axiom M6}$$

$$\mathsf{mult}^{\mathsf{M}}_X\ (\mathsf{unit}^{\mathsf{M}}_{\mathsf{M}X}\ y) = y \tag{Axiom M7}$$

$$\mathsf{mult}^{\mathsf{M}}_X\ (\mathsf{lift}^{\mathsf{M}}_{X,\mathsf{M}X}\ \mathsf{unit}^{\mathsf{M}}_X\ y) = y \tag{Axiom M8}$$

$$\mathsf{mult}^{\mathsf{M}}_X\ (\mathsf{mult}^{\mathsf{M}}_{\mathsf{M}X}\ z) = \mathsf{mult}^{\mathsf{M}}_X\ (\mathsf{lift}^{\mathsf{M}}_{\mathsf{M}(\mathsf{M}X),\mathsf{M}X}\ \mathsf{mult}^{\mathsf{M}}_X\ z) \tag{Axiom M9}$$

for all $X, Y : \mathsf{Type}, f : X \to Y, x : X, y : \mathsf{M}\,(\mathsf{M}\,X)$ and $z : \mathsf{M}\,(\mathsf{M}\,(\mathsf{M}\,X))$.

A term of the nondeterministic type $\mathsf{M}X$ is regarded as the result of a nondeterministic computation in X. Under a belief that our type theory has to be effective enough to do classical reasoning on the possible return values of the nondeterministic computation, we suggest a characterization of the monad by relating it with the classical nonempty power-set monad that we can construct within the type theory:

$$\mathsf{P}_+X :\equiv \Sigma(S : X \to \mathsf{Prop}).\ \exists(x : X).\ S\,x$$

We can confirm that the coherence conditions for the monad hold with function lift, unit, and multiplication:

$$\begin{aligned}
\mathsf{lift}^{\mathsf{P}_+}_{X,Y} f &:\equiv \lambda((S, -)).\ \big(\lambda(y : Y).\ \exists(z : X).\ (y = f\,z) \times (S\,z), -\big) \\
\mathsf{unit}^{\mathsf{P}_+}_X\ x &:\equiv (\lambda(y : X).\ x = y, -) \\
\mathsf{mult}^{\mathsf{P}_+}_X\ x &:\equiv \big(\lambda(y : X).\ \Sigma(z : \mathsf{P}_+X).\ (\pi_1\,z\,y) \times (\pi_1\,x\,z), -\big)
\end{aligned}$$

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

Here, π_1 is the first projection of Σ-types and the occurrences of $-$ represent some classical proof terms.

In order to relate it with the classical nonempty power-set monad, we assume that there is a submonoidal natural transformation

$$\mathsf{picture} : \Pi(X : \mathsf{Type}).\ \mathsf{M}X \to \mathsf{P}_+X. \tag{Axiom M10}$$

It is a submonoidal natural transformation in that (i) it is a natural transformation

$$\Pi(X, Y : \mathsf{Type}).\ \Pi(f : X \to Y).\ (\mathsf{lift}^{\mathsf{P}_+}_{X,Y} f) \circ \mathsf{picture}_X = \mathsf{picture}_Y \circ (\mathsf{lift}^{\mathsf{M}}_{X,Y} f), \tag{Axiom M11}$$

(ii) for any $X : \mathsf{Type}$, $\mathsf{picture}_X$ is monic

$$\Pi(x : X).\ \Pi(y : X).\ \mathsf{picture}_X x = \mathsf{picture}_X y \to x = y, \tag{Axiom M12}$$

and (iii) the coherence conditions which on the unit is

$$\mathsf{picture}_X \circ \mathsf{unit}^{\mathsf{M}}_X = \mathsf{unit}^{\mathsf{P}_+}_X \tag{Axiom M13}$$

and on the multiplication is

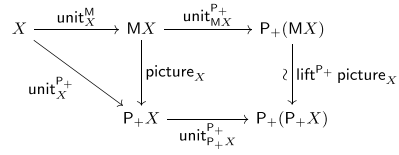
$$\mathsf{picture}_X \circ \mathsf{mult}^{\mathsf{M}}_X = \mathsf{mult}^{\mathsf{P}_+}_X \circ \mathsf{picture}_{\mathsf{P}_+X} \circ (\mathsf{lift}^{\mathsf{M}} \mathsf{picture}_X) \tag{Axiom M14}$$

hold. In other words, thinking of it in the setting of category theory, we assume that our monad M is a submonoidal object of P_+ in the category of endofunctors.

We further characterize the nondeterminism by that the classically lifted picture $\mathsf{lift}^{\mathsf{P}_+}\ \mathsf{picture}_X : \mathsf{P}_+(\mathsf{M}X) \to \mathsf{P}_+(\mathsf{P}_+X)$ is a natural isomorphism that admits an inverse:

$$\Pi(X : \mathsf{Type}).\ \mathsf{is_equiv}\ (\mathsf{lift}^{\mathsf{P}_+}_{\mathsf{M}X,\mathsf{P}_+X} \mathsf{picture}_X) \tag{Axiom M15}$$

The following diagram shows the relation between the two monads:



When $x : \mathsf{M}X$ is a nondeterministic element, the natural transformation can be regarded as a classical picture containing the elements x represents and can be used to reason on them. However, if the purpose is to refer to the elements of $x : \mathsf{M}X$, we often do not need the second part of $\mathsf{picture}_X\, x$, which is about the nonemptiness of x. Hence, let us make the definition

$$\mathsf{pic}_X : \mathsf{M}X \to (X \to \mathsf{Prop}) :\equiv \lambda(x : \mathsf{M}X).\ \pi_1(\mathsf{picture}_X x)$$

so that we can conveniently express $\mathsf{pic}_X\ x\ y : \mathsf{Prop}$ to say that $y : X$ is a possible outcome of $x : \mathsf{M}X$.

The last building block in relating the two monads is a destruction method:

$$\Pi(X : \mathsf{Type}).\ \mathsf{M}X \to \mathsf{M}\Sigma(y : X).\ \mathsf{pic}_X\, x\, y. \tag{Axiom M16}$$

Namely, when we have a nondeterministic object x, we can nondeterministically get a pair (y, t) where $y : X$ and t is a reason why y can be nondeterministically obtained from x.

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

We introduce some additional axioms to characterize nondeterministic computation. For any two semi-decidable decisions $x, y : \mathsf{K}$, if promised that either of x or y holds classically, we can nondeterministically decide whether x holds or y holds:

$$\mathsf{select} : \Pi(x, y : \mathsf{K}).\ (\lceil x \rceil \vee \lceil y \rceil) \to \mathsf{M}\big(\lceil x \rceil + \lceil y \rceil\big)\,. \qquad \text{(Axiom M17)}$$

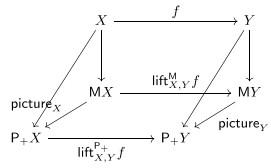
If a type X is sub-singleton, we can eliminate the nondeterminism on $\mathsf{M}X$:

$$\mathsf{elimM} : \Pi(X : \mathsf{Type}).\ (\Pi(x, y : X).\ x = y) \to \mathsf{is_equiv}_{X,\mathsf{M}X}\ \mathsf{unit}^{\mathsf{M}}_X. \qquad \text{(Axiom M18)}$$

Here, $\mathsf{is_equiv}_{X,Y} f$ a type saying that two types X, Y are equivalent by $f : X \to Y$, is defined by $\Sigma(g : Y \to X).\ (\Pi(x : X).\ g\,(f\,x) = x) \wedge \Pi(y : Y).\ f\,(g\,y) = y$ as our type theory is extensional.

EXAMPLE 1.
For any proposition P, suppose both $\mathsf{semiDec}(P)$ and $\mathsf{semiDec}(\neg P)$ hold. As $P \vee \neg P$ holds by the classical law of excluded middle, we have $\mathsf{M}(P + \neg P)$ by applying select. As it is provable that $P + \neg P$ is sub-singleton, using elimM, we have $P + \neg P$, the decidability of the proposition P.

EXAMPLE 2.
An example of using $\mathsf{picture}$ is when we analyse the values of nondeterministically lifted functions. See the commutative diagram:



When we have a lifted function $\mathsf{lift}^{\mathsf{M}}_{X,Y} f$ and a nondeterministic object $x : \mathsf{M}X$, we see the nondeterministic output $\mathsf{lift}^{\mathsf{M}}_{X,Y} f\ x$ by taking its picture $\mathsf{picture}_Y\ (\mathsf{lift}^{\mathsf{M}}_{X,Y} f\ x)$. Directly from the assumptions of $\mathsf{picture}$, propositional extensionality, and functional extensionality, we can easily show that

$$\mathsf{pic}_Y\ (\mathsf{lift}^{\mathsf{M}}_{X,Y} f\ x) = \lambda(z : Y).\ \exists(y : X).\ \mathsf{pic}_X x\ y \wedge z = f\ y$$

holds. In words, $z : Y$ is a possible output of $(\mathsf{lift}^{\mathsf{M}}_{X,Y} f\ x)$ if and only if there classically is $y : X$ such that y is a possible output of $x : \mathsf{M}X$ and z is $f\ y$.

The equation can be used in the following implicational forms:

$$\mathsf{pic}_X x\ y \to \mathsf{pic}_Y(\mathsf{lift}^{\mathsf{M}}_{X,Y} f\ x)\ (f\ y)$$

$$\mathsf{pic}_Y(\mathsf{lift}^{\mathsf{M}}_{X,Y} f\ x)\ z \to \exists(y : X).\ \mathsf{pic}_X x\ y \wedge z = f\ y$$

3.3 Nondeterministic dependent choice

Suppose any sequence of types $P : \mathsf{N} \to \mathsf{Type}$ and a nondeterministic procedure that runs through the types $f : \Pi(n : \mathsf{N}).\ (P\,n) \to \mathsf{M}(P\,(n+1))$. We can think of a procedure of repeatedly and indefinitely applying the nondeterministic procedure: e.g. $f_n(\cdots f_2(f_1(f_0\,x_0))\cdots)$ where $x_0 : P\,0$. Though the expression is not well-typed, intuitively, from the computational point of view, when we

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

apply it repeatedly, we nondeterministically get a sequence that selects through $P\,n$. Starting from x_0, we get nondeterministically $x_1 : P\,0$ from $f\,0\,x_0 : \mathsf{M}\,(P\,0)$. Then, according to the nondeterministic choice $x_1 : P\,0$ amongst $f\,0\,x_0 : \mathsf{M}\,(P\,0)$, we again get nondeterministically $x_2 : P\,1$ from $f\,1\,x_1 : \mathsf{M}\,(P\,1)$. Repeating this forever, we get a specific (nondeterministic) sequence where each entry depends on the nondeterministic choices that have been made in the previous entries.

One may think the primitive recursion of natural numbers, which already exists in the base type theory, and which is meant to express repeated applications, will do the job. The primitive recursion of natural numbers applied to the type family P is of type

$$\mathsf{M}\,(P\,0) \to (\Pi(n : \mathsf{N}).\, \mathsf{M}\,(P\,n) \to \mathsf{M}\,(P\,(n+1))) \to \Pi(n : \mathsf{N}).\, \mathsf{M}\,(P\,n).$$

Given $f : \Pi(n : \mathsf{N}).\, (P\,n) \to \mathsf{M}(P\,(n+1))$ and $x_0 : P\,0$, applying the recursion on $\mathsf{unit}^{\mathsf{M}}_{P\,0} x_0$ and $\lambda(n : \mathsf{N}).\, \lambda(x : \mathsf{M}(P\,n)).\, \mathsf{mult}^{\mathsf{M}}(\mathsf{lift}^{\mathsf{M}}(f\,n)\,x)$ denotes exactly applying f repeatedly on x_0. However, the result of the application does not preserve any information on the dependency between the sequential nondeterministic choices as we can see that the result is of type $\Pi(n : \mathsf{N}).\, \mathsf{M}(P\,n)$.

For example, let us consider $P\,n :\equiv \mathsf{N}$ and

$$f\,n\,x :\equiv \begin{cases} 0 \text{ or } 1 & \text{if } n = 0, \\ x & \text{otherwise.} \end{cases}$$

When we repeatedly apply the procedure on 0, we expect to have one of the two sequences $0, 0, 0, 0, 0, \cdots$ or $0, 1, 1, 1, 1, \cdots$ nondeterministically. However, when we apply the primitive recursion, all we can get is the sequence of the nondeterministic real numbers $0, (0$ *textor* $1), (0$ or $1), \cdots$ which is less informative, forgetting all the information about the dependencies that f creates. Hence, we need a separate and more expressive principle but with computational behaviour identical to primitive recursion.

Suppose any sequence of types $P : \mathsf{N} \to \mathsf{Type}$ and a sequence of classical binary relations $R : \Pi(n : \mathsf{N}).\, P\,n \to P\,(n+1) \to \mathsf{Prop}$. The binary relation is where the dependencies between sequential choices are encoded. For the above example, $R\,n\,x\,y$ can be set to $n > 0 \to x = y$. We call a function of type

$$\Pi(n : \mathsf{N}).\, \Pi(x : P\,n).\, \mathsf{M}\Sigma(y : P\,(n+1)).\, R\,n\,x\,y$$

an M-*trace* of R. Note that admitting a trace automatically ensures that R is a (classically) entire relation.

We introduce a new principle called nondeterministic dependent choice in our system. The name nondeterministic dependent choice comes from the observation that when applying the principle with the double negation monad or the propositional truncation monad (assuming they are provided by the type theory), the principle becomes the classical dependent choice and intuitionistic dependent choice, respectively.

The nondeterministic dependent choice (M-dependent choice for short) says that for any M-trace of R, there is a term of type

$$\mathsf{M}\Sigma(g : \Pi(n : \mathsf{N}).\, P\,n).\, \Pi(m : \mathsf{N}).\, R\,m\,(g\,m)\,(g\,(m+1))$$

satisfying a coherence condition that will be described below. In words: From a trace of R, we can nondeterministically get a sequence g that runs through R.

Given any M-trace f of R, now there are two different ways of constructing a term of type $\Pi(n : \mathsf{N}).\, \mathsf{M}(P\,n)$, forgetting the information on the dependencies. The first is to naively apply the primitive recursion on f as described in the beginning of this subsection. The second is to apply the following

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

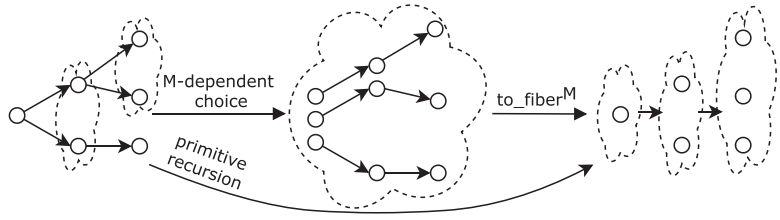


FIGURE 2. Intuitive picture on the coherence condition for the M-dependent choice.

operation:

$$\mathsf{to_fiber}^{\mathsf{M}}(g : \mathsf{M}\Pi(n : \mathsf{N}).\, P\,n) :\equiv \lambda(n : \mathsf{N}).\, \left(\mathsf{lift}^{\mathsf{M}}(\lambda(h : \Pi(n : \mathsf{N}).\, P\,n).\, h\,n)\, g\right)$$

on the M-lifted first projection of the M-dependent choice. The coherence condition states that the two operations of forgetting the information on the paths are identical (cf. Fig. 2).

our type theory admits M *– dependent choice.* (Axiom M19)

In [27], we axiomatized that there is a term constant $\omega\mathsf{lift}$ such that for any $P : \mathsf{N} \to \mathsf{Type}$, it holds that $\omega\mathsf{lift}\,P : (\Pi(n : \mathsf{N}).\, \mathsf{M}\,(P\,n)) \to \mathsf{M}\Pi(n : \mathsf{N}).\, P\,n$ is a section of $\mathsf{to_\ fiber}^{\mathsf{M}}$. In other words, for any $f : \Pi(n : \mathsf{N}).\, \mathsf{M}\,(P\,n)$, it holds that $\mathsf{to_\ fiber}\,(\omega\mathsf{lift}\,f) = f$. From a computational point of view, it says that when we have a sequence of nondeterministic computations, we can nondeterministically choose one sequence of (deterministic) computations. The countable lift is derivable from the nondetermistic dependent choice.

LEMMA 1.
The countable lift property $\omega\mathsf{lift}$ is derivable.

PROOF. Suppose any sequence of types $P : \mathsf{N} \to \mathsf{Type}$ and a sequence of nondeterministic objects $f : \Pi(n : \mathsf{N}).\, \mathsf{M}(P\,n)$. Then, we can set $R\,n\,x\,y :\equiv \mathsf{True}$ and verify that the function

$$\lambda(n : \mathsf{N}).\, \lambda(x : P\,n).\, (f\,(n+1), \star)$$

is a trace where $\star : \mathsf{True}$. Hence, applying the M-lifted dependent choice, we can get a term of type $\mathsf{M}\Pi(n : \mathsf{N}).\, P\,n$. □

3.4 Nondeterministic logic

When we have a nondeterministic object $x : \mathsf{M}X$, regarding it as the result of some nondeterministic computation, it is desirable to analyse properties of the possible outcomes of the nondeterministic computation x. For example, when we have a nondeterministic real number $x : \mathsf{MR}$, we might want to know if every possible outcomes of x is nonzero or if there is at least one possible outcome that is zero. Generally speaking, for a classical predicate $P : X \to \mathsf{Prop}$ and a nondeterministic object $x : \mathsf{M}X$, it is desirable to have a way to express a proposition saying *all possible outcomes* $y : X$ *of* x *satisfies* $P\,y$ or *some possible outcomes* $y : X$ *of* x *satisfies* $P\,y$.

The most simple way to achieve this is to use the natural transformation $\mathsf{picture}_X : \mathsf{M}X \to \mathsf{P}_+X$ and work with the classical description of $x : \mathsf{M}X$. For example, when we have $P : X \to \mathsf{Prop}$, the proposition that all outcomes of x satisfying P can be described by type $\Pi(y : X).\, \mathsf{pic}_X\,x\,y \to P\,y$. Similarly, the proposition that some outcomes of x satisfying P can be described by type $\exists(y : X).\, \mathsf{pic}_X\,x\,y \times P\,y$. However, the characterization of the nondeterminism allows another formalization. When we have $P : X \to \mathsf{Prop}$, we can naturally lift it to $\mathsf{lift}^{\mathsf{M}}P : \mathsf{M}X \to \mathsf{MProp}$.

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

Hence, if there are retractions of $\mathsf{unit}^{\mathsf{M}}_{\mathsf{Prop}} : \mathsf{Prop} \to \mathsf{MProp}$, we can postcompose them to get naturally defined predicates of type $\mathsf{M}X \to \mathsf{Prop}$.

LEMMA 2.
Define the two operators:

$$\wedge^{\mathsf{M}} : \mathsf{MProp} \to \mathsf{Prop} :\equiv \lambda(x : \mathsf{MProp}).\ x = \mathsf{unit}^{\mathsf{M}}\mathsf{true}$$

$$\vee^{\mathsf{M}} : \mathsf{MProp} \to \mathsf{Prop} :\equiv \lambda(x : \mathsf{MProp}).\ \neg(x = \mathsf{unit}^{\mathsf{M}}\mathsf{false})$$

They are retractions of $\mathsf{unit}^{\mathsf{M}}_{\mathsf{Prop}} : \mathsf{Prop} \to \mathsf{MProp}$. In other words, for any $P : \mathsf{Prop}$, it holds that $\wedge^{\mathsf{M}}(\mathsf{unit}^{\mathsf{M}}_{\mathsf{Prop}}\, P) = P$ and $\vee^{\mathsf{M}}(\mathsf{unit}^{\mathsf{M}}_{\mathsf{Prop}}\, P) = P$ hold.

PROOF. To prove that $\wedge^{\mathsf{M}}$ is a retract, we need to show that $\wedge^{\mathsf{M}}(\mathsf{unit}^{\mathsf{M}}_{\mathsf{Prop}} P) = P$ holds for any P. As the target, which is an identity type, is in Prop, we can do branching on $P \vee \neg P$ using Axiom TT1. Further using the classical propositional extensionality (Axiom TT2), we only need to show that $\wedge^{\mathsf{M}}(\mathsf{unit}^{\mathsf{M}}_{\mathsf{Prop}}\mathsf{True}) = \mathsf{True}$ and $\wedge^{\mathsf{M}}(\mathsf{unit}^{\mathsf{M}}_{\mathsf{Prop}}\mathsf{False}) = \mathsf{False}$ hold.

Unfolding the definition of $\wedge^{\mathsf{M}}$, it means to prove

$$\left(\mathsf{unit}^{\mathsf{M}}_{\mathsf{Prop}}\mathsf{True} = \mathsf{unit}^{\mathsf{M}}_{\mathsf{Prop}}\mathsf{True}\right) = \mathsf{True}$$

and

$$\left(\mathsf{unit}^{\mathsf{M}}_{\mathsf{Prop}}\mathsf{False} = \mathsf{unit}^{\mathsf{M}}_{\mathsf{Prop}}\mathsf{True}\right) = \mathsf{False}$$

which are obvious from the classical propositional extensionality and the fact that $\mathsf{unit}^{\mathsf{M}}$ is monic. The fact that $\mathsf{unit}^{\mathsf{M}}$ is monic is direct from the fact that $\mathsf{picture}$ and $\mathsf{unit}^{\mathsf{P}_+}$ are monic.

The other case of $\vee^{\mathsf{M}}$ can be proven similarly. □

Let us for a moment depart from the type theory and think in the classical setting where we consider Prop as a set $\{\top, \bot\}$. Then, MProp can be regarded as $\{\{\top, \bot\}, \{\top\}, \{\bot\}\}$. Hence, classically, there can be exactly two retractions of the unit that is decided by whether we map $\{\top, \bot\}$ to $\top$ or to $\bot$. It is not too difficult to see $\wedge^{\mathsf{M}}$ is a retraction that maps $\{\top, \bot\}$ to $\bot$ and $\vee^{\mathsf{M}}$ is a retraction that maps $\{\top, \bot\}$ to $\top$, justifying the notations.

DEFINITION 1.
For any type X, a classical predicate $P : X \to \mathsf{Prop}$, and a nondeterministic object $x : \mathsf{M}X$, let us define

$$\forall^{\mathsf{M}} x\, P :\equiv (\wedge^{\mathsf{M}} \circ \mathsf{lift}^{\mathsf{M}} P)\ x \quad \text{and} \quad \exists^{\mathsf{M}} x\, P :\equiv (\vee^{\mathsf{M}} \circ \mathsf{lift}^{\mathsf{M}} P)\ x.$$

Let us write $\forall^{\mathsf{M}} y : x.\ P(y)$ for $\forall^{\mathsf{M}} x\, (\lambda(y : X).\ P(y))$ and $\exists^{\mathsf{M}} y : x.\ P(y)$ for $\exists^{\mathsf{M}} x\, (\lambda(y : X).\ P(y))$. Furthermore, for any $x : \mathsf{M}X$ and $y : X$, define

$$y \in^{\mathsf{M}} x :\equiv \exists^{\mathsf{M}} z : x.\ y = z.$$

The following lemma supports that the axiomatization of $\mathsf{picture}$ works as intended.

LEMMA 3.
Within the type theory, for any type X, a classical predicate $P : X \to \mathsf{Prop}$, and a nondeterministic object $x : \mathsf{M}X$, the following equations hold:

$$\forall^{\mathsf{M}} x\, P = \Pi(y : X).\ \mathsf{pic}_X\, x\, y \to P\, y \ \text{ and } \ \exists^{\mathsf{M}} x\, P = \exists(y : X).\ \mathsf{pic}_X\, x\, y \wedge P\, y.$$

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

Furthermore, for any $y : X$, it holds that

$$y \in^{\mathsf{M}} x = \mathsf{pic}_X\, x\, y.$$

PROOF. For the first equation, using the classical propositional extensionality, we only need to prove that $(\forall^{\mathsf{M}} x\, P) \to (\Pi(y : X).\ \mathsf{pic}_X\, x\, y \to P\, y)$ and $(\Pi(y : X).\ \mathsf{pic}_X\, x\, y \to P\, y) \to (\forall^{\mathsf{M}} x\, P)$ hold. Unfolding the definitions, the implications become

$$\left(\mathsf{lift}^{\mathsf{M}}_{X,\mathsf{Prop}} P\, x = \mathsf{unit}^{\mathsf{M}}_{\mathsf{Prop}} \mathsf{True}\right) \to (\Pi(y : X).\ \mathsf{pic}_X\, x\, y \to P\, y) \tag{1}$$

and

$$(\Pi(y : X).\ \mathsf{pic}_X\, x\, y \to P\, y) \to \left(\mathsf{lift}^{\mathsf{M}}_{X,\mathsf{Prop}} P\, x = \mathsf{unit}^{\mathsf{M}}_{\mathsf{Prop}} \mathsf{True}\right). \tag{2}$$

For Equation (4), we need to prove that $P\ y$ holds with the assumptions $\mathsf{lift}^{\mathsf{M}}_{X,\mathsf{Prop}} P\ x = \mathsf{unit}^{\mathsf{M}}_{\mathsf{Prop}} \mathsf{True}$, $y : X$, and $\mathsf{pic}_X x\ y$. From example 2, since we have $\mathsf{pic}_X x\ y$ as an assumption, we can derive $\mathsf{pic}_{\mathsf{Prop}}(\mathsf{lift}^{\mathsf{M}}_{X,\mathsf{Prop}} P\ x)\ (P\ y)$. Thanks to the identity type $\mathsf{lift}^{\mathsf{M}}_{X,\mathsf{Prop}} P\, x = \mathsf{unit}^{\mathsf{M}}_{\mathsf{Prop}} \mathsf{True}$, we get

$$\mathsf{pic}_{\mathsf{Prop}} \left(\mathsf{unit}^{\mathsf{M}}_{\mathsf{Prop}} \mathsf{True}\right)\ (P\, y). \tag{3}$$

Therefore, by Axiom M13 on Equation (6), we get $\mathsf{unit}^{\mathsf{P}_+}_{\mathsf{Prop}} \mathsf{True}\ (P\, y)$. Unfolding the definition of $\mathsf{unit}^{\mathsf{P}_+}$, we can get $P\, y = \mathsf{True}$ which is enough to prove the goal $P\, y$.

For Equation (5), we need to prove the identity $\mathsf{lift}^{\mathsf{M}}_{X,\mathsf{Prop}} P\ x = \mathsf{unit}^{\mathsf{M}}_{\mathsf{Prop}} \mathsf{True}$ assuming $\Pi(y : X).\ \mathsf{pic}_X\, x\, y \to P\, y$. Applying the fact that pic is monic, we can simplify the goal to

$$\mathsf{pic}_{\mathsf{Prop}} \left(\mathsf{lift}^{\mathsf{M}}_{X,\mathsf{Prop}} P\, x\right) = \mathsf{pic}_{\mathsf{Prop}} \left(\mathsf{unit}^{\mathsf{M}}_{\mathsf{Prop}} \mathsf{True}\right)$$

where the right hand side can be reduced to $\lambda(y : \mathsf{Prop}).\ y = \mathsf{True}$ by Axiom M13. Hence, applying the functional extensionality and classical propositional extensionality, the goal gets divided into the two sub-goals:

$$\left(\mathsf{pic}_{\mathsf{Prop}} \left(\mathsf{lift}^{\mathsf{M}}_{X,\mathsf{Prop}} P\, x\right)\ y\right) \to y = \mathsf{True} \tag{4}$$

and

$$\mathsf{pic}_{\mathsf{Prop}} \left(\mathsf{lift}^{\mathsf{M}}_{X,\mathsf{Prop}} P\, x\right)\ \mathsf{True}. \tag{5}$$

For Equation (7), from example 2, we have the equation $\mathsf{pic}_{\mathsf{Prop}} \left(\mathsf{lift}^{\mathsf{M}}_{X,\mathsf{Prop}} P\, x)\ y\right) = \exists(z : X).\ \mathsf{pic}_X x\ z \wedge y = P\ z$. Hence, we need to derive $y = \mathsf{True}$ from $\exists(z : X).\ \mathsf{pic}_X x\ z \wedge y = P\ z$ and $\Pi(y : X).\ \mathsf{pic}_X\, x\, y \to P\, y$.

Since the goal is in Prop, we can instantiate $z : X$ such that $\mathsf{pic}_X x\ z$ and $y = P\ z$ hold. Therefore, from the initial assumption $\Pi(y : X).\ \mathsf{pic}_X\, x\, y \to P\, y$ we have $P\ z$. From classical propositional extensionality, we can prove $P\ z = \mathsf{True}$. Hence, we can derive $y = \mathsf{True}$.

For Equation (8), from example 2, we have the equation $\mathsf{pic}_{\mathsf{Prop}}(\mathsf{lift}^{\mathsf{M}}_{X,\mathsf{Prop}} P\ x)\ \mathsf{True}) = \exists(z : X).\ \mathsf{pic}_X x\ z \wedge \mathsf{True} = P\ z$. Hence, the goal can be rewritten as

$$\exists(z : X).\ \mathsf{pic}_X x\ z \wedge \mathsf{True} = P\ z$$

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

Since the goal, which is in Prop is provably sub-singleton, we can reduce the goal to

$$\mathsf{M}\exists(z : X).\ \mathsf{pic}_X x\, z \wedge \mathsf{True} = P\, z$$

using Axiom M18.

Using Axiom M16, from the assumption that we have $x : \mathsf{M}X$, we have $\mathsf{M}\Sigma(z : X).\ \mathsf{pic}_X x\, z$. Since the goal is M-lifted, we can apply $\mathsf{lift}^{\mathsf{M}}$ to get $\Sigma(z : X).\ \mathsf{pic}_X x\, z$. Hence, we can get $z : X$ such that $\mathsf{pic}_X x\, z$ holds. Now, applying it in the initial assumption $\Pi(y : X).\ \mathsf{pic}_X\, x\, y \to P\, y$, we get $P\, z$. Hence, applying $\mathsf{unit}^{\mathsf{M}}$ with the z being the witness of the $\exists$ type, we get

$$\mathsf{M}\exists(z : X).\ \mathsf{pic}_X x\, z \wedge \mathsf{True} = P\, z.$$

Other equations can be proven similarly. □

In practice, when using our axiomatization, we often get an object of type $\mathsf{M}\Sigma(x : X).\ P\, x$. For example, by constructing a term $f : (A + B) \to \Sigma(x : X).\ P\, x$ using the case distinction on A or B, when we lift it, we get $\mathsf{lift}^{\mathsf{M}} f : \mathsf{M}(A + B) \to \mathsf{M}\Sigma(x : X).\ P\, x$. Then, precomposing an assumption that A and B are nondeterministically choosable, we get a term of type $\mathsf{M}\Sigma(x : X).\ P\, x$ meaning that we can nondeterministically obtain $x : X$ such that $P\, x$ holds. Since now we have a tool to deal with each entries of a nondeterministic object, we can come up with the following lemma that splits M-lifted Σ-types:

LEMMA 4.
For any type $X : \mathsf{Type}$ and a classical predicate $P : X \to \mathsf{Prop}$, it holds that

$$\left(\Sigma(x : \mathsf{M}X).\ \forall^{\mathsf{M}} y : x.\, P\, y\right) \leftrightarrow \mathsf{M}\Sigma(x : X).\ P\, x$$

In words, when we have a nondeterministic existence $\mathsf{M}\Sigma(x : X).\ P\, x$, we can split it to obtain a deterministic existence of a nondeterministic object $x : \mathsf{M}X$ such that every entry of it satisfies P, and vice versa.

4 Real numbers and Limits

Let us now introduce the basic axioms that we use for the real number type. A defining feature of exact real computation is the ability to compute certain limits of user-defined sequences. Its counterpart in the axiomatization of real numbers is the principle of metric completeness.

There are three distinct cases where we need to compute limits:

(1) when a deterministic sequence of real numbers converges to a deterministic point,
(2) when a sequence of nondeterministic real numbers converges to a deterministic point and
(3) when a sequence of nondeterministic real numbers converges to a nondeterministic point.

We deal with each of them separately in Section 4.2 to 4.4.

While the first one, i.e. how the limit computation should be axiomatized in the logical language is well-established [6, 15, 50], it has been under debate how to deal with the case when nondeterminism is involved in the limit computation [44, 24]. Note that this situation occurs quite naturally even for simple operations such as computing square roots of complex numbers (cf. Section 7.4). Early versions of the C++ library iRRAM therefore already provided a simple nondeterministic limit operation as primitive [38,§ 10.3] which the authors of the software recently suggested to replace by a more generic operation for nondeterministic limits [44]. In Section 3.3 we defined *nondeterministic dependent choice*, a simple and natural principle of nondeterminism which we apply in Section 4.4

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

to show that the nondetermistic limit operation can be derived in our theory and therefore does not need to be introduced as a primitive operation.

Finally, we propose some simple extensions of the theory of real numbers to complex numbers and general Euclideean spaces.

4.1 Basic axiomatization of real numbers

We assume real numbers by declaring that there is a type for real numbers containing two distinct constants 0 and 1 and the standard arithmetical operators.

$$\mathsf{R} : \mathsf{Type} \tag{Axiom R1}$$

$$0 : \mathsf{R} \tag{Axiom R2}$$

$$1 : \mathsf{R} \tag{Axiom R3}$$

$$+ : \mathsf{R} \to \mathsf{R} \to \mathsf{R} \tag{Axiom R4}$$

$$\times : \mathsf{R} \to \mathsf{R} \to \mathsf{R} \tag{Axiom R5}$$

$$- : \mathsf{R} \to \mathsf{R} \tag{Axiom R6}$$

$$/ : \Pi(x : \mathsf{R}).\, x \neq 0 \to \mathsf{R} \tag{Axiom R7}$$

We further assume the existence of a semi-decidable comparison operator.

$$< : \mathsf{R} \to \mathsf{R} \to \mathsf{Prop} \tag{Axiom R8}$$

$$\Pi(x, y : \mathsf{R}).\, \mathsf{semiDec}(x < y) \tag{Axiom R9}$$

That is, the order between any two real numbers (as a classical proposition) is semi-decidable.

We assume the properties of the structure classically in a safe way that does not damage constructivity.

$$\Pi(x, y : \mathsf{R}).\, x + y = y + x \tag{Axiom R10}$$

$$\Pi(x, y, z : \mathsf{R}).\, (x + y) + z = x + (y + z) \tag{Axiom R11}$$

$$\Pi(x : \mathsf{R}).\, x + -x = 0 \tag{Axiom R12}$$

$$\Pi(x : \mathsf{R}).\, 0 + x = x \tag{Axiom R13}$$

$$\Pi(x, y : \mathsf{R}).\, x \times y = y \times x \tag{Axiom R14}$$

$$\Pi(x, y, z : \mathsf{R}).\, (x \times y) \times z = x \times (y \times z) \tag{Axiom R15}$$

$$\Pi(x : \mathsf{R}).\, \Pi(p : x \neq 0).\, (/\, x\, p) \times x = 1 \tag{Axiom R16}$$

$$\Pi(x : \mathsf{R}).\, 1 \times x = x \tag{Axiom R17}$$

$$\Pi(x, y, z : \mathsf{R}).\, x \times (y + z) = x \times y + x \times z \tag{Axiom R18}$$

$$1 \neq 0 \tag{Axiom R19}$$

$$1 > 0 \tag{Axiom R20}$$

$$\Pi(x, y : \mathsf{R}).\, x < y \vee x = y \vee x > y \tag{Axiom R21}$$

$$\Pi(x, y : \mathsf{R}).\, x < y \to \neg(y < x) \tag{Axiom R22}$$

$$\Pi(x, z, y : \mathsf{R}).\, x < y \to y < z \to x < z \tag{Axiom R23}$$

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

$$\Pi(x,y,z:\mathsf{R}).\ y<z\to x+y<x+z \tag{Axiom R24}$$

$$\Pi(x,y,z:\mathsf{R}).\ 0<x\to y<z\to x\times y<x\times z \tag{Axiom R25}$$

$$\Pi(x:\mathsf{R}).\ x>0\to\exists(n:\mathsf{N}).\ 2^{-n}<x \tag{Axiom R26}$$

Note that, for example, trichotomy is only assumed classically and an inhabitant of the type $\Pi(x,y:\mathsf{R}).\ (x<y)+(x=y)+(y<x)$ is not posed anywhere.

EXAMPLE 3.
As real comparison is only semi-decidable, in exact real computation it is often replaced by a so-called ϵ-test which roughly says, for any real numbers x,y,ϵ, when ϵ is positive, we can nondeterminstically decide if $x<y+\epsilon$ or $y<x+\epsilon$. This corresponds to the multivalued version of the approximate splitting lemma [48, Lemma 1.23]

$$\mathsf{mSplit}:\Pi(x,y,\epsilon:\mathsf{R}).\ 0<\epsilon\to\mathsf{M}\big((x<y+\epsilon)+(y<x+\epsilon)\big)$$

which is derivable in the theory.

PROOF. Classical trichotomy (Axiom R21) lets us construct the term

$$\Pi(x,y,\epsilon:\mathsf{R}).\ 0<\epsilon\to x<y+\epsilon\vee y<x+\epsilon.$$

Since the inequalities are semi-decidable we can apply select (Axiom M10) on this term to derive mSplit. □

4.2 Deterministic limits

The first case is exactly the ordinary metric completeness, which is realized by the primitive limit operations in exact real number computation software. We introduce it as an axiom in our theory. For a sequence $f:\mathsf{N}\to\mathsf{R}$, we define the property of f being a fast Cauchy sequence by

$$\mathsf{is_Cauchy}f:\equiv\Pi(n,m:\mathsf{N}).\ \Pi(x:fn).\ \Pi(y:fm).\ -2^{-n-m}\le x-y\le 2^{-n-m},$$

and the property of x being a limit of the sequence f by

$$\mathsf{is_limit}\,xf:\equiv\Pi(n:\mathsf{N}).\ \Pi(y:fn).\ -2^{-n}\le x-y\le 2^{-n}.$$

The constructive completeness axiom then says that, for any fast Cauchy sequence, we can construct its limit point:

$$\Pi(f:\mathsf{N}\to\mathsf{R}).\ \mathsf{is_Cauchy}f\to\Sigma(x:\mathsf{R}).\ \mathsf{is_limit}\,xf \tag{Axiom R27}$$

Often, we already have some classical description P of a real number x and want to show that we can constructively get the said number. The following variation of the limit can thus be useful:

LEMMA 5.
For any predicate $P:\mathsf{R}\to\mathsf{Prop}$, we can construct the term

$$\big(\exists!\,(z:\mathsf{R}).\ Pz\big)\to\big(\Pi(n:\mathsf{N}).\ \Sigma(e:\mathsf{R}).\exists(a:\mathsf{R}).\ Pa\wedge|e-a|\le 2^{-n}\big)\to\Sigma(a:\mathsf{R}).\ Pa.$$

PROOF. The sequence defined in the premise of the statement can be shown to be a fast Cauchy sequence and thus by Axiom R27 we can construct its limit point a. Also, using the Archimedan Axiom (Axiom R26), we can conclude that the limit point is unique. Thus, Pa holds. □

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

The absolute value function $|x|$ in the above lemma is defined by taking the maximum of x and $-x$. The maximization function is constructed using the limit operation. The construction is illustrated in detail later in Section 7.1. However, when we use the absolute values of the form $|x| \leq y$ for some $x, y : \mathsf{R}$, since the type is in Prop, using classical propositional extensionality we can prove that it is identical to $-y \leq x \leq y$.

REMARK 1.
Classical completeness states that any sequence that is classically Cauchy (i.e. without the additional assumption of a guaranteed speed of convergence), classically admits a limit point, or, equivalently, any classically nonempty and bounded subset of real numbers classically admits a least upper bound. The classical completeness is weaker and can be derived from constructive completeness using our set of axioms.

4.3 Deterministic limits of nondeterministic sequences

Now consider the case where there is a single real number we want to obtain and we have a nondeterministic procedure approximating said number.

Suppose we have a nondeterministic sequence $f : \mathsf{N} \to \mathsf{MR}$. We can extend the notion of being a Cauchy sequence and a limit point to nondeterministic sequences by

$$\mathsf{is_Cauchy}^{\mathsf{M}} f :\equiv \Pi(n, m : \mathsf{N}).\ \Pi^{\mathsf{M}}(x : f\, n).\ \Pi^{\mathsf{M}}(y : f\, m).\ |x - y| \leq 2^{-n-m},$$

and

$$\mathsf{is_limit}^{\mathsf{M}}\, x f :\equiv \Pi(n : \mathsf{N}).\ \Pi^{\mathsf{M}}(y : f\, n).\ |x - y| \leq 2^{-n}.$$

We can prove that for any nondeterministic Cauchy sequence, there deterministically and constructively exists the limit.

THEOREM 1.
Within our type theory, we can construct a term of the type

$$\Pi(f : \mathsf{N} \to \mathsf{MR}).\ \mathsf{is_Cauchy}^{\mathsf{M}} f \to \Sigma(x : \mathsf{R}).\ \mathsf{is_limit}^{\mathsf{M}}\, x f.$$

In words, a nondeterministic sequence converges to a point if all possible candidates of the nondeterministic sequence converge to the point.

PROOF. Given $f : \mathsf{N} \to \mathsf{MR}$ with $\mathsf{is_\ Cauchy}^{\mathsf{M}} f$, we first prove that the limit point is unique, i.e. for any $x, y : \mathsf{R}$

$$\mathsf{is_limit}^{\mathsf{M}}\, x f \to \mathsf{is_limit}^{\mathsf{M}}\, y f \to x = y.$$

Thus, by Axiom M18 it suffices to show that

$$\mathsf{M}\Sigma(x : \mathsf{R}).\ \mathsf{is_limit}^{\mathsf{M}}\, x f.$$

We can nondeterministically choose a sequence through f, i.e. we can show

$$\mathsf{M}\Sigma(s : \mathsf{N} \to \mathsf{R}).\ \Pi(n : \mathsf{N}).\ s\, n \in^{\mathsf{M}} f\, n.$$

By Axiom M4 it suffices to show

$$\left(\Sigma(s : \mathsf{N} \to \mathsf{R}).\ \Pi(n : \mathsf{N}).\ s\, n \in^{\mathsf{M}} f\, n\right) \to \Sigma(x : \mathsf{R}).\ \mathsf{is_limit}^{\mathsf{M}}\, x f.$$

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

As the sequence s is a fast Cauchy sequence it has a limit point x by Axiom R27 and we can show $\mathsf{is_limit}^{\mathsf{M}}\, x\, f$. □

As for the deterministic case we often already have a classical description of real numbers that we want to construct. For example, when we compute a square root of a real number $x : \mathsf{R}$, we first define it classically by $S : \mathsf{R} \to \mathsf{Prop} :\equiv \lambda(y : \mathsf{R}).\, x = y \times y$ then prove $\Sigma(y : \mathsf{R}).\, S\, y$.

For any real number $x : \mathsf{R}$ and a classical description of real numbers $S : \mathsf{R} \to \mathsf{Prop}$, define the notation: $x \sim_n S :\equiv \exists(y : \mathsf{R}).\, (S\, y) \times |x - y| \leq 2^{-n}$ saying that x approximates a real number represented by S by 2^{-n}. Then, we can derive the following version of metric completeness:

COROLLARY 1.
We can derive the term

$$\big(\exists!\, (x : \mathsf{R}).\, S\, x\big) \to \big(\Pi(n : \mathsf{N}).\, \mathsf{M}\Sigma(y : \mathsf{R}).\, y \sim_n S\big) \to \Sigma(y : \mathsf{R}).\, S\, y.$$

4.4 Nondeterministic limits

Suppose we are given a classical description of real numbers $S : \mathsf{R} \to \mathsf{Prop}$ that is classically sequentially closed, i.e. we define $\mathsf{is_seq_closed}\, S$ to be the following type:

$$\Pi(f : \mathsf{N} \to \mathsf{R}).\, (\Pi(n : \mathsf{N}).\, (f\, n) \sim_n S) \to \exists(x : \mathsf{R}).\, (S\, x) \times \mathsf{is_limit}\, x\, f.$$

A *nondeterministic refinement procedure* is a procedure that for each natural number n and real number x_n with a promise $x_n \sim_n S$ nondeterministically computes a 2^{-n-1} approximation to some real number in S which is at most 2^{-n-1} apart from x_n. Note that x_n and x_{n+1} do not necessarily approximate the same number in S.

Formally, we define a nondeterministic refinement procedure as a function f of type

$$f : \Pi(n : \mathsf{N}).\, \Pi(x : \mathsf{R}).\, x \sim_n S \to \mathsf{M}\Sigma(y : \mathsf{R}).\, \big(|x - y| \leq 2^{-n-1}\big) \times \big(y \sim_{n+1} S\big).$$

We show that given such a nondeterministic refinement procedure, we can apply the M-dependent choice to nondeterministically get a point in S. We call such a point the limit point of the procedure.

THEOREM 2.
We can construct a term of type

$$\begin{aligned}
&\big(\Pi(S : \mathsf{R} \to \mathsf{Prop}).\, \mathsf{is_seq_closed}\, S\big) \to \\
&\quad \big(\mathsf{M}\Sigma(y : \mathsf{R}).\, y \sim_0 S\big) \to \\
&\quad \big(\Pi(n : \mathsf{N}).\, \Pi(x : \mathsf{R}).\, x \sim_n S \to \mathsf{M}\Sigma(y : \mathsf{R}).\, \big(|x - y| \leq 2^{-n-1}\big) \times \big(y \sim_{n+1} S\big)\big) \to \\
&\quad \mathsf{M}\Sigma(y : \mathsf{R}).\, S\, y
\end{aligned}$$

PROOF. Let

$$P\, n :\equiv \Sigma(x : \mathsf{R}).\, x \sim_n S \quad \text{and} \quad R\, n\, x\, y :\equiv |\pi_1\, x - \pi_1\, y| \leq 2^{-n-1}.$$

Any refinement procedure f can be easily adjusted to become an M-trace of R. The M-dependent choice on f with an initial approximation $x_0 : \mathsf{M}\Sigma(y : \mathsf{R}).\, y \sim_n S$, yields a sequence $g : \mathsf{N} \to \mathsf{R}$ that is consecutively close, i.e.

$$\Pi(n : \mathsf{N}).\, |(g\, n) - (g\, (n + 1))| \leq 2^{-n-1},$$

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

and converges to an element in S, i.e. $\Pi(n : \mathsf{N}).\ g\,n \sim_n S$. As S is sequentially closed and we can prove that g is Cauchy, applying the ordinary limit on S constructively yields a point in S. Hence, applying $\mathsf{lift}^{\mathsf{M}}$ on the procedure and postcomposing it to the result of the M-dependent choice yields the nondeterministic limit. □

The nondeterministic refinement procedure defined above is quite generic, and it is often difficult to define such a procedure. Specifically, it requires to consecutively refine any possible previous approximation to a better approximation of a limit. In practice, the previous approximation will not be arbitrary, but already derived according to the rules of the procedure. It is therefore useful to think of cases where all possible approximations throughout the indefinite refinement procedure share some invariant properties.

To make this more clear, let us again consider the nondeterministic function

$$f\,n\,x :\equiv \begin{cases} 0 \text{ or } 1 & \text{if } n = 0, \\ x & \text{otherwise,} \end{cases}$$

similar to the one from Section 3.3.

Starting with $1/2$, the function nondeterministically generates the two sequences $1/2, 0, 0, \cdots$ and $1/2, 1, 1 \cdots$. Both are Cauchy sequences that converge to 0 and 1, respectively, thus we would consider 0 and 1 possible limit points. However, note that f is not a refinement procedure in the previous sense: When 2^{-n} is given as a 2^{-n} approximation to $0, f$ returns 2^{-n}, which is not a 2^{-n-1} approximation to any of 0 or 1. In other words, we lose the information that when iterating f, we only encounter either 0 or 1 when $n > 0$. Thus, to apply Theorem 2 we would have to artificially modify the procedure to deal with arbitrary approximations when $n > 0$. Instead, we would like to use the invariant property of f directly to build a more effective nondeterministic limit operation.

Let $S : \mathsf{R} \to \mathsf{Prop}$ be a classical description of real numbers that is sequentially closed. We declare an invariant property of approximations $Q : \mathsf{N} \to \mathsf{R} \to \mathsf{Type}$ that is preserved throughout the refinements. We can embed the information of Q into P at the step of applying the M-dependent choice:

$$P\,n :\equiv \Sigma(x : \mathsf{R}).\ (x \sim_n S) \times Q\,n\,x.$$

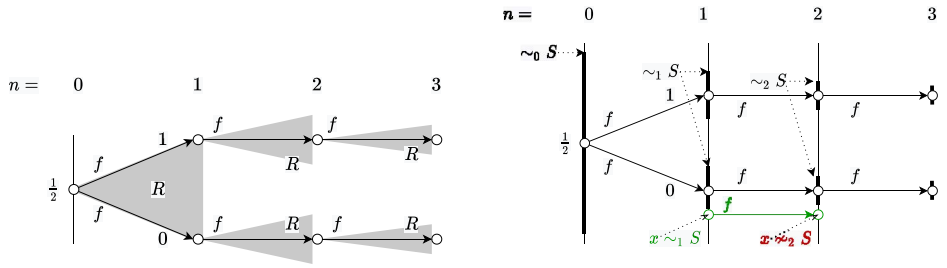
A similar derivation as in the previous section yields the following more informative limit operation:

THEOREM 3.
Within our type theory, we can construct a term of type

$$\begin{aligned} &\big(\Pi(S : \mathsf{R} \to \mathsf{Prop}).\ \Pi(Q : \mathsf{N} \to \mathsf{R} \to \mathsf{Type}).\ \mathsf{is_seq_closed}\,S\big) \to \\ &\quad \big(\mathsf{M}\Sigma(y : \mathsf{R}).\ (y \sim_0 S) \times Q\,0\,y\big) \to \\ &\quad \big(\Pi(n : \mathsf{N}).\ \Pi(x : \mathsf{R}).\ (x \sim_n S) \times Q\,n\,x \to \\ &\qquad \mathsf{M}\Sigma(y : \mathsf{R}).\ (|x - y| \le 2^{-n-1}) \times (y \sim_{n+1} S) \times (Q\,(n+1)\,y)\big) \to \\ &\quad \mathsf{M}\Sigma(y : \mathsf{R}).\ S\,y \end{aligned}$$

Note that the required nondeterministic refinement procedure accepts additional information on its input $Q\,n\,x : \mathsf{Type}$, on which we can do effective reasoning as it is indexed through Type. For example, in the above case of 0 and 1, we can let $Q\,n\,x :\equiv n > 0 \to (x = 0) + (x = 1)$ such that in

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025



(a) An M-*trace* of $R\,n\,x\,y \equiv |x - y| \leq 2^{-n}$ converging to $S = \{0, 1\}$

(b) Without the invariant Q, the refinement procedure f is not admissible.

FIGURE 3. Using an invariant property of f to define a limit.

the beginning of each refinement step $n > 0$, we can effectively test if x is 0 or 1. The price to pay is that in each step we have to construct a Boolean term which indicates whether the refinement is 0 or 1, which then is used in the next refinement step. Figure 3 illustrates this example.

REMARK 2.
As mentioned above, the iRRAM C++ framework provides a similar operation `limit_mv`. The operator computes the limit of a nondeterministic sequence using an additional discrete hint `choice` that restricts the possible values of the limit [37].

4.5 Complex numbers and general Euclidean spaces

Having defined real numbers, it is natural to think of possible extensions such as complex numbers or real vector spaces. Adding complex numbers to our theory is simple. We define a complex number as a pair of real numbers, i.e. $\mathsf{C} :\equiv \mathsf{R} \times \mathsf{R}$ and define the field operations in the standard way.

We further define a maximum norm $|\cdot| : \mathsf{C} \to \mathsf{R}$ and a distance $d\ z_1\ z_2 :\equiv |z_1 - z_2|$. For any complex number, we can separate the real and imaginary parts and treat them like ordinary real numbers to prove some of their properties. Specifically, we can extend all the limit operators defined in this section to complex numbers. The use of the maximum norm for the distance makes the proofs straightforward.

More generally, for any $n\colon \mathsf{N}$, we can define the type $\mathsf{euclidean}_n$ simply as an n-element list of R with a maximum norm, and define the vector space operations point-wise.

5 Soundness by realizability

To prove soundness of the set of axioms that we present, we extend the standard realizability interpretation that is introduced in Section 2.2. That means we need to declare the interpretations of our axiomatic types that are assemblies and the interpretations of axiomatic terms that are computable functions. Then, the interpretations of complex types and terms are defined inductively following the original construction of the interpretation. For example, if we define the interpretation of R, the interpretation of $\mathsf{R} \to \mathsf{R}$ is defined automatically as the exponential of the interpretation of R.

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

We interpret K, of course, as the Kleenean assembly $\mathbf{K}$ (Axiom K1). The axiomatic term constant true is interpreted as the computable point $tt : \mathbf{1} \to \mathbf{K}$ (Axiom K2) and false is interpreted as the computable point $ff : \mathbf{1} \to \mathbf{K}$ (Axiom K3). See that the interpretation of the type $\mathsf{true} \neq \mathsf{false}$ is $\mathbf{1}$, which is trivially inhabited. Hence, we confirm the validity of Axiom K4.

Similarly, we interpret the algebraic operations in Axiom K5 to Axiom K7 to the corresponding computable operations of Kleeneans $\mathbf{K}$ in Fig. 1. See that the related classical properties Axiom K8 to Axiom K10 get interpreted to the final assembly $\mathbf{1}$. For the last axiom Axiom K11, see that the interpretation of the type $\Pi(x : \mathsf{K}).\, x \downarrow\!\to \lceil x \rceil + \lfloor x \rfloor$ is the assembly of functions that decides if $x = tt$ or $x = ff$ given that $x \neq \bot$. Such a function is realizable by a machine that simply iterates through any given realizer of x checking if it encounters 1 or 2, as it is ensured that the realizer is not $0^{\mathbb{N}}$, which is the only realizer of $\bot$.

We interpret R as any effective assembly of reals $\mathbf{R}$ (Axiom R1). We map $0, 1 : \mathsf{R}$ to the computable points $0, 1 : \mathbf{1} \to \mathsf{R}$ (Axiom R2 and R3) and map $+, \times, -$ to the corresponding computable functions in $\mathsf{Asm}(\mathcal{K}_2)$ seen in Fact 1. One subtle part is the multiplicative inversion (Axiom R7) which is axiomatized in our type theory as

$$/ : \Pi(x : \mathsf{R}).\, x \neq 0 \to \mathsf{R}.$$

See that the interpretation of the type is an assembly of the dependent functions:

$$f : \mathsf{R} \ni x \mapsto \begin{cases} |\mathbf{R^0}| & \text{if } x = 0, \\ |\mathbf{R^1}| & \text{otherwise,} \end{cases}$$

which gets defined by nominating a partial function $\mathbb{R} \setminus \{0\} \to \mathbb{R}$. Due to the triviality of a function from the empty-set, a realizer of such f is allowed to diverge when it receives a realizer of $0 \in \mathbb{R}$. In our case, we of course interpret the division to the multiplicative inversion that is not defined at $0 \in \mathbb{R}$.

We interpret $<: \mathsf{R} \to \mathsf{R} \to \mathsf{Prop}$ as the following function:

$$x \llbracket < \rrbracket y :\equiv \begin{cases} \mathbf{1} & \text{if } x < y, \\ \mathbf{0} & \text{otherwise} \end{cases}$$

which is trivially computable because the realization relation on Prop is trivial (Axiom R8). Under this interpretation, we can validate Axiom R9 which is direct from that the order relation of real numbers is semi-decidable.

All the classical properties Axioms R9 to R26 are validated that their interpretations get to be $\mathbf{1}$, which requires some clerical works to verify. The validity of the deterministic limit axiom Axiom R27 is also direct from the computability of limits of fast Cauchy sequences that are seen from Section 2.1.

In order to interpret the nondeterminism monad M, we specify an endofunctor $\mathbf{M} : \mathsf{Asm}(\mathcal{K}_2) \to \mathsf{Asm}(\mathcal{K}_2)$ in our model. For an assembly $\mathbf{X}$, $\mathbf{M}(\mathbf{X})$ is an assembly of the set of nonempty subsets of $\mathbf{X}$ whose realization relation $\Vdash_{\mathbf{M}(\mathbf{X})}$ is defined by

$$\varphi \Vdash_{\mathbf{M}(\mathbf{X})} S \quad :\Leftrightarrow \quad \exists x.\, x \in S \wedge \varphi \Vdash_{\mathbf{X}} x.$$

In words, φ realizes a nonempty subset S of $\mathbf{X}$ if φ realizes an element x of S in the original $\mathbf{X}$. Observe that for any assemblies $\mathbf{X}, \mathbf{Y}$, a multivalued function $f : \mathbf{X} \rightrightarrows \mathbf{Y}$ is computable in the sense of computable analysis [54] if and only if it appears as a morphism $f : \mathbf{X} \to \mathbf{M}(\mathbf{Y})$.

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

The endofunctor $\mathbf{M}$ can be easily proven to be a monad with the unit $\eta_{\mathbf{X}} : x \mapsto \{x\}$, the multiplication $\mu_{\mathbf{X}} : S \mapsto \bigcup_{T \in S} T$, and the action on morphisms $\mathbf{M}(f) : S \mapsto \bigcup_{x \in S} \{f(x)\}$. The computability of the functions hold trivially.

We interpret our nondeterminism type former $\mathsf{M} : \mathsf{Type} \to \mathsf{Type}$ as the monad (Axiom M1) $\mathbf{M} : \mathsf{Asm}(\mathcal{K}_2) \to \mathsf{Asm}(\mathcal{K}_2)$ and interpret the unit (Axiom M2), the multiplication (Axiom M3), and the function lift (Axiom M4) appropriately as the computable functions of the monad structure. Then, the types of the coherence conditions, Axioms M5–M9, get interpreted to $\mathbf{1}$.

To validate the core characterization of the nondeterminism, which is based on the relation with the classical nonempty power-set monad P_+, let us see the interpretation of the monad in the model. See that for any type X, it holds that

$$|[\![\mathsf{P}_+(X)]\!]| = \{(S : [\![X]\!] \to \{\mathbf{1}, \mathbf{0}\}, \star) \mid \exists x.\, S\, x = \mathbf{1}\}$$

where any φ realizes any $(S, \star)$ as any $S : [\![X]\!] \to [\![\mathsf{Prop}]\!]$ is trivially computable.

Calling this induced monad $\mathbf{P}_+ : \mathsf{Asm}(\mathcal{K}_2) \to \mathsf{Asm}(\mathcal{K}_2)$, see that the underlying set of $\mathbf{P}_+(\mathbf{X})$ is set-theoretically isomorphic to the underlying set of $\mathbf{M}(\mathbf{X})$ for any assembly $\mathbf{X}$. Further, for any assembly $\mathbf{X}$, the assembly $\mathbf{P}_+(\mathbf{X})$ has a trivial realization relation. Hence, there is a monic (and set-theoretically surjective) natural transformation defined for each $\mathbf{X}$:

$$\mathbf{picture}_{\mathbf{X}} : \mathbf{M}(\mathbf{X}) \ni S \mapsto (S, \star) \in \mathbf{P}_+(\mathbf{X})$$

which can be trivially computed as $\mathbf{P}_+(\mathbf{X})$ has the trivial realization relation. We interpret the natural transformation $\mathsf{picture}$ (Axiom M10) to the natural transformation **picture**. Other classical properties Axioms M11–M14 can be validated.

Since the underlying sets of the interpretations $\mathsf{P}_+(\mathsf{M}X)$ and $\mathsf{P}_+(\mathsf{P}_+X)$ are set-theoretically isomorphic and they both have trivial realization relation, it can be easily proved that

$$\mathbf{P}_+(\mathbf{picture}_{\mathbf{X}}) : \mathbf{P}_+(\mathbf{M}(\mathbf{X})) \to \mathbf{P}_+(\mathbf{P}_+(\mathbf{X}))$$

is an isormorphism in $\mathsf{Asm}(\mathcal{K}_2)$, for any $\mathbf{X}$, whose inverse is also trivially computable. This observation validates Axiom M15. The destruction method Axiom M16 becomes obvious under this interpretation.

When we have two Kleeneans $x, y \in \mathbf{K}$, in the relevant case that at least one of them is tt, we can devise a machine that iterates through the given realizers of x and y to check if 1 appears in the given realizer of x or in the given realizer of y. Calling this machine φ, for $\varphi_x \Vdash_{\mathbf{K}} x$ and $\varphi_y \Vdash_{\mathbf{K}} y$, $\varphi(\varphi_x, \varphi_y)$ realizes $\{tt, ff\}$ when $x = y = tt$, $\{tt\}$ when $x = tt \wedge y \neq tt$, and $\{ff\}$ when $x \neq tt \wedge y = tt$ in $\mathbf{M}(\mathbf{K})$. See that this procedure, **select**, which is obviously computable, realizes the interpretation of Axiom M17. When $\mathbf{X}$ is sub-singleton, $\mathbf{M}(\mathbf{X})$ is actually identical to $\mathbf{X}$. Hence, Axiom M18 gets validated. As it was discussed in the section, the last axiom Axiom M19 gets validated by the fact that we can form recursion on the level of realizers.

Discussions thus far conclude the soundness of our axioms:

METATHEOREM 1. The axiomatization is sound admitting a realizability interpretation.

6 Relating classical analysis

As described in the previous sections, our theory allows classical reasoning for noncomputational properties. For example, to prove a statement of the form

$$\Pi(x : \mathsf{R}).\ \Sigma(y : \mathsf{R}).\ P\, x\, y$$

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

where $P : \mathsf{R} \to \mathsf{R} \to \mathsf{Prop}$, we assume any $x : \mathsf{R}$, provide an explicit $y : \mathsf{R}$ and prove that $P\,x\,y$ holds. $P\,x\,y : \mathsf{Prop}$ is a classical statement and thus we do not need to restrict ourselves to constructive reasoning in its proof. In terms of program extraction (cf. Section 8), the computational content of the statement is just a program that maps any $x : \mathsf{R}$ to a $y : \mathsf{R}$ and the information that $P\,x\,y$ holds is not part of the program itself.

In a sense we can thus often separate a proof in a constructive part, which we would like to map to a program in a certain way, and a purely mathematical part, where we are mostly concerned about correctness and care less about the exact form of the proof. Now, this poses the practical question on how much of classical mathematics needs to be re-proven over our real number type R.

Of course, there are already several formalizations of the classical real numbers in proof assistants, which provide many of the statements we need. Instead of starting from scratch, we would like to access such results and be able to apply them in our proofs.

To make this precise, let us assume we have a second real number type $\tilde{\mathsf{R}}$ and we have already proven some results over $\tilde{\mathsf{R}}$. Let us further assume that different from our formalization, $\tilde{\mathsf{R}}$ is completely classical, by which we mean that classical statements such as

$$\Pi(x : \tilde{\mathsf{R}}).\, x > 0 + \neg(x > 0)$$

hold. Readers familiar with the Coq proof assistant can for example think of the real number type in the Coq standard library. We want to embed our axiomatization and apply theorems proven over the classical theory to our formalization while separating the constructive part and the classical part of the type theory correctly so that realizability results like those from Section 5 still hold.

Note that, even though now the type theory provides classical types and terms through $\tilde{\mathsf{R}}$, it should stay fully constructive for the terms that are constructed without making use of any of the classical axioms. We can thus think of a term in the type theory as to be formally interpreted in two different models.

Let us define the two type judgements

$$\models t : X$$

saying that t of type X may rely on classical axioms and

$$\vdash t' : X'$$

saying that t' of type X' is defined without any classical axioms.

When $\models t : X$, we interpret it in the category of sets Set and when $\vdash t : X$, we interpret it in $\mathsf{Asm}(\mathcal{K}_2)$.

EXAMPLE 4.
There is a term t such that

$$\models t : \Pi(x : \tilde{\mathsf{R}}).\, x > 0 + \neg(x > 0)$$

is derivable but

$$\vdash t : \Pi(x : \tilde{\mathsf{R}}).\, x > 0 + \neg(x > 0)$$

does not hold.

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

Our goal is to relate the two type judgements. As we do not assume any counter-classical axioms such as the continuity principle in our theory, every constructively proven statement is also true classically. Thus, obviously when $\vdash t : X$ is derivable, so is $\models t : X$.

However, we are interested in the other direction, i.e. how to get a constructively well-typed term from a classically well-typed term. Set is a reflective subcategory of $\mathsf{Asm}(\mathcal{K}_2)$ by the forgetful functor $\boldsymbol{\Gamma} : \mathsf{Asm}(\mathcal{K}_2) \to \mathsf{Set}$ and its right adjoint $\nabla : \mathsf{Set} \to \mathsf{Asm}(\mathcal{K}_2)$, where for any set X, ∇X is the assembly of X with the trivial realization relation [53, Theorem 1.5.2].

For each type X, define

$$\nabla X :\equiv \Sigma(P : X \to \mathsf{Prop}).\ \exists!\,(x : X).\ P\,x\,.$$

See that for any type X, $[\![\models \nabla X : \mathsf{Type}]\!]$ is isomorphic to $[\![\models X : \mathsf{Type}]\!]$ in Set and $[\![\vdash \nabla X : \mathsf{Type}]\!]$ is isomorphic to $\nabla \boldsymbol{\Gamma} [\![\vdash X : \mathsf{Type}]\!]$ in $\mathsf{Asm}(\mathcal{K}_2)$. It can be understood as a functor that erases all the computational structure of X while keeping its set-theoretic structure.

It is provable in the type theory using the assumptions of Prop being the type of classical propositions admitting propositional extensionality that ∇ is an idempotent monad where its unit $\mathsf{unit}_\nabla : \Pi(X : \mathsf{Type}).\ X \to \nabla X$ on ∇X, is an equivalence with the inverse being the multiplication. Moreover, it holds that $\mathsf{unit}_\nabla\,\mathsf{Prop} : \mathsf{Prop} \to \nabla\mathsf{Prop}$ is an equivalence. That means, given a mapping $f : X_1 \to X_2 \to \cdots \to X_d$, there is a naturally defined lifting $f^{\dagger\nabla} \colon \nabla X_1 \to \nabla X_2 \to \cdots \to \nabla X_d$ and given a predicate $P : X_1 \to X_2 \to \cdots \to \mathsf{Prop}$, there is $P^{\dagger\nabla} \colon \nabla X_1 \to \nabla X_2 \to \cdots \to \mathsf{Prop}$.

We add the type judgement rule:

$$\frac{\models t : X}{\vdash t : X}\ X \text{ is transferable (RELATE)}$$

saying that when t is a classically constructed term of a transferable type X, we have a constructive term t of type X. A type is transferable if it is of the following form:

1. ∇c for a constant type, or a Type or Prop variable c,
2. $X \to Y, X \times Y, X \wedge Y, X \vee Y$ or $\nabla(X + Y)$ for transferable types X and Y,
3. $\Pi(x : X).\ P(x)$, $\Sigma(x : X).\ P(x)$ or $\exists(x : X).\ P(x)$ for transferable types X and $P(x)$,
4. $x = y, x < y$ or $x <^\dagger y$, or
5. Type or Prop.

Roughly speaking, a type is transferable if its subexpressions in the construction of the type are guarded by ∇.

This judgement rule is validated in our interpretation. First, note that $\nabla\{*\} \simeq \mathbf{1}$. When a type X is transferable, $[\![\vdash X : \mathsf{Type}]\!] \simeq \nabla[\![\models X : \mathsf{Type}]\!]$ holds. When $\models t : X$, we have a function $[\![\models t : X]\!] : \{*\} \to [\![\models X : \mathsf{Type}]\!]$ in Set. Hence, we define $[\![\vdash t : X]\!] : \mathbf{1} \to [\![\vdash X : \mathsf{Type}]\!]$ by pre and postcomposing the above isomorphisms to $\nabla[\![\models t : X]\!] : \nabla\{*\} \to \nabla[\![\models X : \mathsf{Type}]\!]$.

To relate our axiomatic real numbers with the classical real number type $\tilde{\mathsf{R}}$, we assume the existence of a map $\mathsf{relator}$ with interpretation in Set being the identity map $[\![\models \mathsf{relator} : \mathsf{R} \to \nabla\tilde{\mathsf{R}}]\!] : \mathbb{R} \ni x \mapsto x \in \mathbb{R}$.

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

We assume the following axioms that characterize the mapping.

$$\mathsf{relator} : \mathsf{R} \to \nabla\tilde{\mathsf{R}} \qquad \text{(Axiom } \nabla 1\text{)}$$

$$\Pi(x,y : \mathsf{R}).\ \mathsf{relator}\, x = \mathsf{relator}\, y \to x = y \qquad \text{(Axiom } \nabla 2\text{)}$$

$$\Pi(y : \nabla\tilde{\mathsf{R}}).\ \exists(x : \mathsf{R}).\ y = \mathsf{relator}\, x \qquad \text{(Axiom } \nabla 3\text{)}$$

$$\mathsf{relator}\ 0 = \mathsf{unit}_\nabla\, \tilde{\mathsf{R}}\ 0 \qquad \text{(Axiom } \nabla 4\text{)}$$

$$\mathsf{relator}\ 1 = \mathsf{unit}_\nabla\, \tilde{\mathsf{R}}\ 1 \qquad \text{(Axiom } \nabla 5\text{)}$$

$$\Pi(x,y : \mathsf{R}).\ \mathsf{relator}\ (x+y) = (\mathsf{relator}\, x) +^{\dagger\nabla} (\mathsf{relator}\, y) \qquad \text{(Axiom } \nabla 6\text{)}$$

$$\Pi(x,y : \mathsf{R}).\ \mathsf{relator}\ (x\times y) = (\mathsf{relator}\, x) \times^{\dagger\nabla} (\mathsf{relator}\, y) \qquad \text{(Axiom } \nabla 7\text{)}$$

$$\Pi(x : \mathsf{R}).\ \mathsf{relator}\ (-x) = -^{\dagger\nabla}\ (\mathsf{relator}\, x) \qquad \text{(Axiom } \nabla 8\text{)}$$

$$\Pi(x : \mathsf{R}).\ \Pi(p : x \neq 0).\ \mathsf{relator}\ (/\,x\,p) = /^{\dagger\nabla}\ (\mathsf{relator}\, x) \qquad \text{(Axiom } \nabla 9\text{)}$$

$$\Pi(x,y : \mathsf{R}).\ (x < y) = (\mathsf{relator}\, x) <^{\dagger\nabla} (\mathsf{relator}\, y) \qquad \text{(Axiom } \nabla 10\text{)}$$

EXAMPLE 5.
Using relator, we can prove the classical properties of R without using the classical axioms of R. For example, suppose we want to prove

$$\Pi(x,y : \mathsf{R}).\ x + y = y + x.$$

Using the classical real numbers, we can trivially prove

$$\vDash \mathsf{comm}_{\tilde{\mathsf{R}}} : \Pi(x,y : \nabla\tilde{\mathsf{R}}).\ x +^{\dagger\nabla} y = y +^{\dagger\nabla} x\ .$$

Since $x,y : \tilde{\mathsf{R}} \vDash x +^{\dagger\nabla} y : \nabla\tilde{\mathsf{R}}$, from the added judgement rule, we have

$$x,y : \tilde{\mathsf{R}} \vdash x +^{\dagger\nabla} y : \nabla\tilde{\mathsf{R}}.$$

Therefore, the above proof $\mathsf{comm}_{\tilde{\mathsf{R}}}$ directly transfers to a constructive proof

$$\vdash \mathsf{comm}_{\tilde{\mathsf{R}}} : \Pi(x,y : \nabla\tilde{\mathsf{R}}).\ x +^{\dagger\nabla} y = y +^{\dagger\nabla} x.$$

Similarly, from the order decidability of the classical real,

$$\Pi(x : \tilde{\mathsf{R}}).\ (x > 0) + (x = 0) + (x < 0)$$

we can lift it to obtain

$$\Pi(x : \nabla\tilde{\mathsf{R}}).\ (x > 0) \vee (x = 0) \vee (x < 0)$$

Now, using the relator, we can obtain the classical total order

$$\Pi(x : \mathsf{R}).\ (x > 0) \vee (x = 0) \vee (x < 0)$$

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

We will look at a more practical example in Section 7.3.

Note that in Coq we can not formally deal with having two independent type theories simultaneously and therefore a complete correctness proof when applying the relator notion can only be formulated on the meta-level. We plan to address this issue in future work e.g. by writing a Coq plugin that would allow this distinction.

7 Examples

Let us illustrate the use of our formalization by applications to some standard examples from computable analysis.

7.1 Maximization

A simple example of an operation that requires multivaluedness in its definition is the maximization operator that takes two real numbers x and y and returns their maximum. That is, we define

$$\mathsf{max}\, x\, y\, z :\equiv (x > y \to z = x) \wedge (x = y \to z = x) \wedge (x < y \to z = y)$$

and want to derive the term

$$\Pi(x\, y : \mathsf{R}).\ \Sigma(z : \mathsf{R}).\ \mathsf{max}\, x\, y\, z.$$

Partiality of comparisons prevents a direct proof of the statement. Instead, we apply the limit operator defined in corollary 1. That is, we need to show that there is exactly one $z : \mathsf{R}$ for which $\mathsf{max}\, x\, y\, z$ holds and that

$$\Pi(n : \mathsf{N}).\ \mathsf{M}\Sigma(z' : \mathsf{R}).\ z' \sim_n z.$$

The first statement follows from the classical trichotomy. For the second one, we use multivalued branching (example 3) to construct the nondeterministic sequence

$$m_{x,y}\, n :\equiv \begin{cases} x & \text{if } x > y - 2^{-n}, \\ y & \text{if } y > x - 2^{-n}. \end{cases}$$

That is, we construct the approximation by concurrently testing whether $x > y - 2^{-n}$, in which case x is a good enough approximation for the maximum, or $x < y + 2^{-n}$, in which case y is a good enough approximation.

7.2 Intermediate value theorem

A classical example from computable analysis (see e.g. [54, Chapter 6.3]) is finding the zero of a continuous, real valued function $f : [0, 1] \to \mathbb{R}$ with $f(0) < 0$ and $f(1) > 0$ under the assumption that there is exactly one zero in the interval (i.e. a constructive version of the intermediate value theorem from analysis).

We can define continuity of a function $f : \mathsf{R} \to \mathsf{R}$ using the usual ϵ-δ-criterion

$$\mathsf{continuous}\, f :\equiv \Pi(x : \mathsf{R}).\ \Pi(\epsilon > 0).\ \exists(\delta > 0).\ \Pi(y : \mathsf{R}).\ |x - y| \to |(f\ x) - (f\ y)| < \epsilon$$

and define the property of f having exactly one zero in the interval $[a, b]$ by

$$\mathsf{uniq}\, f\, a\, b :\equiv a < b \wedge f\ a < 0 \wedge f\ b > 0 \wedge \exists!\,(z : \mathsf{R}).\ a < z < b \wedge f\ z = 0.$$

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

Here $\Pi(\epsilon > 0).\, P$ is short for $\Pi(\epsilon : \mathsf{R}).\, \epsilon > 0 \to P$ and $\exists(\epsilon > 0).\, P$ is short for $\exists(\epsilon : \mathsf{R}).\, \epsilon > 0 \wedge P$. Then the property we want to show is

$$\mathsf{IVP} : \Pi(f : \mathsf{R} \to \mathsf{R}).\, \mathsf{continuous}\, f \to \mathsf{uniq}\, f\ 0\ 1 \to \Sigma(z : \mathsf{R}).\, 0 < z < 1 \wedge f\, z = 0.$$

The statement can be proven using the trisection method which is similar to the classical bisection method but avoids uncomputable comparison to 0. That is we inductively define sequences a_i, b_i with $f(a_i) \times f(b_i) < 0$ and $b_i - a_i \leq (2/3)^i$. In each step we let $a'_i :\equiv (2a_i + b_i)/3$, $b'_i :\equiv (a_i + 2b_i)/3$ and in parallel check if $f(a'_i) \times f(b_i) < 0$ or $f(a_i) \times f(b'_i) < 0$. In the first case we set $a_{i+1} :\equiv a'_i$, $b_{i+1} :\equiv b_i$, in the second case $a_{i+1} :\equiv a_i$, $b_{i+1} :\equiv b'_i$. As at least one of the inequalities is true by the assumptions, this selection can be done using the select operator from Section 3. The zero can then be defined using the limit operator.

7.3 Classical proofs and a fast square root algorithm

Let us now look at an example on how the $\mathsf{relator}$ operator from Section 6 can be used. Again, suppose we introduced a type $\tilde{\mathsf{R}}$ of classical real numbers in our theory that already provides proofs of some classical facts about real numbers. In particular, suppose we have a term *sq* saying that for any positive real number, there is a square root:

$$\vDash sq : \Pi(x : \tilde{\mathsf{R}}).\, 0 < x \to \Sigma(y : \tilde{\mathsf{R}}).\, x = y \times y$$

As $\mathsf{unit}_\nabla A : A \to \nabla A$ is an equivalence in the classical type theory, we can derive

$$\vDash sq' : \Pi(x : \nabla\tilde{\mathsf{R}}).\, (\mathsf{unit}_\nabla\, 0) <^{\dagger\nabla} x \to \Sigma(y : \nabla\tilde{\mathsf{R}}).\, x = y \times^{\dagger\nabla} y$$

As the type of the above judgement is transferable, we have

$$\vdash sq' : \Pi(x : \nabla\tilde{\mathsf{R}}).\, (\mathsf{unit}_\nabla\, 0) <^{\dagger\nabla} x \to \Sigma(y : \nabla\tilde{\mathsf{R}}).\, x = y \times^{\dagger\nabla} y$$

Using the axioms of the relator, we can obtain a term of type

$$\vdash \Pi(x : \mathsf{R}).\, 0 < x \to \exists(y : \mathsf{R}).\, x = y \times y : \mathsf{Prop}.$$

This illustrates how we can transport a classical proof of the existence of square root based on $\tilde{\mathsf{R}}$ to a constructive proof of the classical existence of square root based on R.

Let us assume that our classical theory already contains all facts about the square root that we need and that similar to the above example we can transport the classical proofs into our constructive setting. We use the transported results to verify a constructive version of the real square root function that can be used for efficient computation of the square root, thus we only need to focus on the constructive parts of the below proof.

By the real square root of x, we mean the unique nonnegative number y such that $x = y \times y$, and use $y = \sqrt{x}$ as shorthand notation for the property $x = y \times y \wedge y \geq 0$. That is, we want to show

$$\mathsf{rsqrt} : \Pi(x : \mathsf{R}).\, x \geq 0 \to \Sigma(y : \mathsf{R}).\, y = \sqrt{x}$$

The idea on how to construct this y is fairly simple and has already been used e.g. as an example in [25]. The main steps are as follows:

1. We define a restricted version of rsqrt in the interval $[0.25, 2]$ using the well-known Heron method.

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

2. For any number $x > 0$, we can find a $z \in \mathbb{Z}$ such that $4^z x \in [0.25, 2]$, i.e. we can scale the number to be inside the interval, apply the restricted square root and rescale the result accordingly.
3. We extend to $x \geq 0$ by approximating the square root of small enough numbers by 0.

Let us start with the first item, i.e. we show

$$\Pi(x : \mathsf{R}).\ \frac{1}{4} \leq x \leq 2 \to \Sigma(y : \mathsf{R}).\ y = \sqrt{x}.$$

We apply constructive completeness (R27), thus we need to show the classical fact that there is exactly one such number $\sqrt{x}$ and

$$\Pi(n : \mathsf{N}).\ \Sigma(y_n : \mathsf{R}).\ \exists(y : \mathsf{R}).\ y = \sqrt{x} \wedge |y_n - y| < 2^{-n}. \tag{6}$$

We derive the former from our classical theory. For the latter we define the sequence

$$\mathsf{heron}\ x\ 0 :\equiv 1$$

$$\mathsf{heron}\ x\ (n+1) :\equiv \frac{1}{2} \times \left(\mathsf{heron}\ x\ n + \frac{x}{\mathsf{heron}\ x\ n}\right).$$

It is well-known that the Heron method converges quadratically to $\sqrt{x}$ in the interval $[0.25, 2]$, i.e. $\left|\mathsf{heron}\ x\ n - \sqrt{x}\right| \leq 2^{-2^n}$ which can be used to show (9).

For the second step, i.e. extending to all positive reals, we first show

$$\Pi(x : \mathsf{R}).\ \mathsf{M}\Sigma(z : \mathsf{Z}).\ \frac{1}{4} \leq (4^z \times x) \leq 2$$

The existence of such a z basically follows from the Archimedean axiom and the Markov principle, but note that we can only find such a z nondeterministically. As $y = \sqrt{4^z \times x} \to 2^{-z} \times y = \sqrt{x}$, we can show

$$\Pi(x : \mathsf{R}).\ x > 0 \to \mathsf{M}(\Sigma(y : \mathsf{R}).\ y = \sqrt{x})$$

and by the uniqueness of the square root Axiom M18 lets us remove the nondeterminism.

Finally, for the last step of extending to 0, we use the limit operation from corollary 1. That is, we need to show

$$\Pi(n : \mathsf{N}).\ \mathsf{M}\Sigma(y_n : \mathsf{R}).\ \exists(y : \mathsf{R}).\ y = \sqrt{x} \wedge |y_n - y| \leq 2^{-n}.$$

Given $n : \mathsf{N}$, we can nondeterminstically choose one of $x < 2^{-2n}$ or $x > 0$. In the first case, we can choose 0 as a 2^{-n} approximation to the square root. In the second case, we can get an exact square root from step 2.

REMARK 3.
Steps 2 and 3 can also be combined in a single step leading to a slightly more efficient algorithm, but we separated the two parts for the sake of presentation.

7.4 Nondeterministic computation of complex square roots

When trying to extend the previous example to complex numbers, a difficulty arises that the square root operation on complex numbers is inherently multivalued. The square root of a number $z \in \mathbb{C}$ is a number $x \in \mathbb{C}$ such that $x^2 = z$. If x is a square root of z, then so is $-x$ and there are no other

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

square roots. Thus, for every $z \neq 0$, there are exactly two square roots. In the case of the square root on nonnegative reals in the previous section we could simply choose one of the two square roots to get a singlevalued branch. However, it is well known that no such continuous choice exists for the whole complex plane: The square root has a branch point at $z = 0$ and thus there is no singlevalued, continuous square root function in any region containing $z = 0$ as an interior point. We show that we can, however, prove the nondeterministic existence of a square root constructively in our theory. That is, we prove the constructive and nondeterministic existence of square roots of complex numbers using a simple method described e.g. in [37].

The following well-known algebraic formula can be used to reduce the calculation of complex square roots to calculating real square roots (see e.g. [17,§6]).

Let $z = a + ib$, then

$$\sqrt{\frac{\sqrt{a^2+b^2}+a}{2}} + i \cdot \operatorname{sgn}(b) \cdot \sqrt{\frac{\sqrt{a^2+b^2}-a}{2}}$$

is one of the square roots of z. Of course, this function is not computable as sgn is not continuous in 0. However, if $z \neq 0$, we can nondeterministically choose one of the cases $a < 0$, $a > 0$, $b < 0$, $b > 0$ and apply the formula. (In case $a > 0$ or $a < 0$, a slight adaption of the formula using $\operatorname{sgn}(a)$ instead of $\operatorname{sgn}(b)$ is used.)

Thus, using the real square root we can show the following restricted version of the existence of a complex square root

$$\surd_0 : \Pi(z : \mathsf{C}).\, z \neq 0 \to \mathsf{M}(\Sigma(x : \mathsf{C}).\, x \times x = z) \tag{7}$$

Finally, we apply Theorem 3 to also include the case $z = 0$. Recall that given a 2^{-n} approximation x_n of a square root of z that satisfies a certain predicate Q that we will define later, we need to choose a $2^{-(n+1)}$ approximation x_{n+1} of a square root of z with $|x_{n+1} - x_n| \leq 2^{n+1}$ and such that x_{n+1} satisfies Q. We proceed as follows. In the beginning, at each step n we nondeterministically choose one of the two cases $|z| < 2^{-2(n+2)}$ or $|z| > 0$. In the first case, 0 is a good enough approximation for any square root of z. In the second case, we know $z \neq 0$ and thus can apply (10) to get the exact value of a square root. However, once we have selected the second case, for any later elements of the sequence we just return the previous value x_n. Thus, all possible sequences the refinement procedure returns have the form $0, 0, 0, \ldots, 0, x, x, x, \ldots$, where x is a square root of z. Further, if we returned 0 at the nth step, we know that $|z| < 2^{-2(n+2)}$ and therefore for any square root x, $|x| < 2^{-(n+2)}$ and returning x at step $n + 1$ is a valid refinement of the previous approximation.

Thus, the invariant property of the sequence defined in this way is given by the relation $Q\, n\, x$: Type defined by

$$Q\, n\, x :\equiv \big((|z| \leq 2^{-2(n+2)}) \times (x = 0)\big) + (x \times x = z).$$

Applying Theorem 3 with this Q, we get

$$\surd : \Pi(z : \mathsf{C}).\, \mathsf{M}(\Sigma(x : \mathsf{C}).\, x \times x = z)$$

7.5 Equivalence of axiomatic real numbers

To prove that the set of axioms we devised to express exact real number computation is expressive enough, we prove that any two types R_1 and R_2 satisfying the set of axioms are type-theoretically equivalent. As our type theory is extensional, they are equivalent if we can construct the mutually

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

inverse functions $\iota_1 : \mathsf{R}_1 \to \mathsf{R}_2$ and $\iota_2 : \mathsf{R}_2 \to \mathsf{R}_1$. The basic idea of the construction is similar to [20] where an effective model-theoretic structure of real numbers is suggested.

From the classical Archimedean principle of real numbers, for any $x : \mathsf{R}_1$, there classically is $z : \mathsf{Z}$ which bounds the magnitude of x in the sense that $|x| < z$ holds. Applying nondeterministically the Markov principle, we can construct the nondeterministic rounding operator:

$$\mathsf{round} : \Pi(x : \mathsf{R}_1).\, \mathsf{M}\Sigma(z : \mathsf{Z}).\, z - 1 < x < z + 1.$$

Recall that the usual rounding is not computable due to discontinuity of the classical rounding function [54, Theorem 4.3.1]. Then, by scaling, we can construct a term of type

$$\mathsf{dyadic} : \Pi(x : \mathsf{R}_1).\, \Pi(n : \mathsf{N}).\, \mathsf{M}\Sigma(z : \mathsf{Z}).\, |x - z \times 2^{-n}| \leq 2^{-n}$$

which nondeterministically approximates the binary magnitude of real numbers.

By using the destruction principle of the nondeterminism and doing some clerical work, we get the fact that for any real number $x : \mathsf{R}_1$, there exists a sequence of nondeterministic integers $f : \mathsf{N} \to \mathsf{MZ}$ such that every section $g : \mathsf{N} \to \mathsf{Z}$ of f is an approximation sequence of x in the sense that $\mathsf{is_limit}\, x\, (\lambda(n : \mathsf{N}).\, (g\, n) \times 2^{-n})$ holds.

Note that the description thus far implicitly used the integer embedding in R_1. Taking out the embedding explicitly, we can prove that for any sequence of integers $g : \mathsf{N} \to \mathsf{Z}$, if its induced dyadic sequence is Cauchy in one type of real numbers, it also is Cauchy in the other one. Hence, from f, using the other integer embedding $\mathsf{Z} \to \mathsf{R}_2$, we can get a sequence of nondeterministic real numbers in R_2 where every section is a Cauchy sequence in R_2. Thus, after proving that the limit points of such sequences is unique, we can apply the deterministic limit of nondeterministic sequences (Theorem 1) to construct a real number in R_2.

Intuitively, we use the space of sequences of nondeterministic integers as an independent stepping stone connecting the two axiomatic types R_1 and R_2. In our axiomatization of nondeterminism we can analyse each section of a sequence of nondeterministic integers so that we can apply a limit operation in R_2.

The other direction $\mathsf{R}_2 \to \mathsf{R}_1$ can be constructed analogously, and the two mappings being inverse to each other can be proved easily, concluding that the two axiomatic types R_1 and R_2 are equivalent.

8 Implementation

We implemented the above theory in the Coq proof assistant[2]. From a correctness proof in our implementation, we can extract Haskell code that uses the AERN library to perform basic real number arithmetic operations. For this, we introduce several extraction rules replacing operations on the constructive reals with the corresponding AERN functions. The extracted code requires only minor mechanical editing, namely adding several import statements (including AERN and comparison operators that return generalized Booleans) and type coercions used in the extracted code to replace erased Coq type translations.

8.1 Overview of the implementation

Our axioms for K M and R are not directly encoded as Coq axioms, but as members of the following type classes:

[2]The source code and detailed usage instructions are on https://github.com/holgerthies/coq-aern/tree/release-2022-01

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

- `LazyBool`: Kleenean operations and their properties, in file `Kleene.v`
- `MultivalueMonad`: operations and properties of M including `choose` (a slightly generalized select) and M-dependent choice, in file `MultivalueMonad.v`
- `ComplArchiSemiDecOrderedField`: operations and properties of R, including limits, in file `RealAxioms.v`

The class `MultivalueMonad` requires also that M is an instance of a type class `Monad` and there exists a homomorphism to the set monad `NPset_Monad` via the type class `Monoid_hom`, formally encoding the 'picture' natural transformation.

We add Coq axioms that declare the existence of models for these type classes:

```
Parameter K : Set.
Axiom K_LazyBool : LazyBool K.

Parameter M : Type -> Type.
Axiom M_Monad : Monad M.
Axiom MultivalueMonad_description : Monoid_hom M_Monad NPset_Monad.
Axiom M_MultivalueMonad :
    @MultivalueMonad _ K_LazyBool _ _ MultivalueMonad_description.

Parameter R : Set.
Axiom R_SemiDecOrderedField : @SemiDecOrderedField  _ K_LazyBool R.
Axiom R_ComplArchiSemiDecOrderedField :
    @ComplArchiSemiDecOrderedField _ _ _ R_SemiDecOrderedField.
```

but we do this only locally, at the point of extracting code. Our application examples do not depend on these axioms, but instead assume the existence of such models in their context. For example, our real maximization implementation includes:

```
Require Import Real.

Section Minmax.
  Generalizable Variables K M Real.

  Context `{klb : LazyBool K} `{M_Monad : Monad M}
          {MultivalueMonad_description : Monoid_hom M_Monad NPset_Monad}
          {M_MultivalueMonad : MultivalueMonad}
          {Real : Type}
          {SemiDecOrderedField_Real : SemiDecOrderedField Real}
          {ComplArchiSemiDecOrderedField_Real :
                                    ComplArchiSemiDecOrderedField}.

Definition real_is_max (x y z : Real)
  := (x > y -> z = x) /\ (x = y -> z = x) /\ (x < y -> z = y).

Lemma real_max_prop : forall x y, {z | real_is_max x y z}.
Proof....                           ComplArchiSemiDecOrderedField}.

        Definition real_is_max (x y z : Real)
          := (x > y -> z = x) /\ (x = y -> z = x) /\ (x < y -> z = y).

        Lemma real_max_prop : forall x y, {z | real_is_max x y z}.
        Proof....
```

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

```
real_max :: CReal -> CReal -> CReal
real_max x y =
  limit $ \(n :: Integer) ->
    let e = 0.5^n in
    if select (x > y - e) (y > x - e)
      then x
      else y
```

FIGURE 4. Real number maximization in Haskell/AERN.

8.2 Exact real computation and the AERN framework

Figure 4 lists a hand-written Haskell/AERN code for computing real number maximization function presented in Sub Section 7.1. This code shows the most important features of AERN. `CReal` is the type of lazy real numbers, essentially a convergent sequence of interval approximations to the real number. The sequence is not necessarily fast converging. Like in iRRAM, the elements of the sequence are computed with increasing precision for dyadic interval endpoints. In practice, the sequences usually converge exponentially (measured in bit-length) since the precision increases exponentially. The usual Haskell Prelude arithmetic operators are being applied to `CReal` numbers in expressions such as 0.5^n and `y - e`.

The real number comparisons `x > y - e` and `y > x - e` do not use the usual Haskell Prelude comparison operators. These comparisons return `CKleenean`, a converging sequence of Kleeneans computed with increasing precisions, exactly like elements of `CReal`. The function `select:: CKleenean -> CKleenean -> Bool` behaves exactly as the select operator defined in Axiom M17. Note the absence of M in the return type of AERN's `select`. Multivaluedness is intrinsic in AERN due to its implementation, as the underlying representation of the Kleenean inputs is redundant.

Finally, the function `limit:: (Integer -> CReal) -> CReal` takes a fast converging sequence of real numbers and returns its limit. In Fig. 4, we have to specify the type of the index `n` since the limit function is in fact generic and can be applied to different types of sequences.

8.3 Code extraction

Haskell/AERN code extraction is defined in file `Extract.v`. The key mappings are listed in Fig. 5. Note that the monad M does not appear in the extracted programs. Multivaluedness is intrinsic thanks to redundancy in the underlying representations. The generalized comparison `Numeric.OrdGenericBool.<` returns the (lazy) Kleenean for real numbers. It is the same comparison that features in Fig. 4.

Figure 6 shows parts of the Coq proof of the maximization operator specification, and relevant parts of the extracted Haskell code.

8.4 Performance measurements

Since our axiomatization of constructive reals is built on a datatype similar to that used by AERN, we expect the performance of the extracted programs to be similar to that of hand-written AERN code.

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

Coq	Haskell
Real	AERN2.CReal
Real0	0
Realplus	(Prelude.+)
limit	AERN2.limit
choose	AERN2.select
Realltb	(Numeric.OrdGenericBool.<)
K	AERN2.CKleenean
sumbool	Prelude.Bool
M	type identity
unitM	Prelude.id
Nat.log2	(integer . integerLog2)

FIGURE 5. Examples of our code extraction mappings.

```
Lemma real_max_prop :
   forall x y, {z | real_is_max x y z}.
Proof.
  intros.
  apply real_mslimit_P_lt.
  + (* max is single valued predicate *)
    ...
  + (* construct limit *)
        intros.
        apply (mjoin (x>y - prec n)
                     (y > x - prec n)).
        ++ intros [c1|c2].
           +++ (* when y-2^n < x *)
         exists x. ...
           +++ (* when x-2^n < y *)
         exists y. ...
        ++ apply M_split.
           apply @prec_pos.
Defined.
```

```
realmax_prop ... x y =
  real_mslimit_P_lt ... (\n ->
    mjoin ...
      (\h ->
        case h of {
          Prelude.True -> x;
          Prelude.False -> y})
      (m_split ... x y (prec ... n)))

m_split ... x y _UU03b5_ =
  choose ...
    (real_lt_semidec ...
      (real_minus ... y _UU03b5_) x)
    (real_lt_semidec ...
      (real_minus ... x _UU03b5_) y)
```

FIGURE 6. Outline of a Coq proof and corresponding extracted Haskell code.

Our measurements, summarized in Fig. 7, are consistent with our hypothesis [3]. iRRAM is known to be one of the most efficient implementations of exact real computation and thus we also included hand-written iRRAM versions for calibration. The last three rows are examples of root finding by trisection. The iRRAM trisection code benefits from in-place update.

Figure 8 shows the execution times of this extracted code on a sample of inputs and with various target precisions. We use logarithmic scales to make the differences easier to see. Slower performance at zero reflects the fact that the limit computation uses the whole sequence, unlike away from zero where only a finite portion of the sequence is evaluated and the faster converging Heron iteration takes over for higher precisions. The closer the input is to zero, the later this switch from limit to Heron takes place when increasing precision. For very large inputs, there is a notable

[3]Benchmarks were run 10 times on a Lenovo T440p laptop with Intel i7-4710MQ CPU and 16GB RAM, OS Ubuntu 18.04, compiled using Haskell Stackage LTS 17.2.

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

Benchmark		Average execution time (s)		
Formula	Accuracy	Extracted	Hand-written	iRRAM
$\max(0, \pi - \pi)$	10^6 bits	16.8	16.2	1.59
$\sqrt{2}$	10^6 bits	0.72	0.72	0.62
$\sqrt{\sqrt{2}}$	10^6 bits	1.51	1.54	1.15
$x - 0.5 = 0$	10^3 bits	3.57	2.3	0.03
$x(2 - x) - 0.5 = 0$	10^3 bits	4.30	3.08	0.04
$\sqrt{x + 0.5} - 1 = 0$	10^3 bits	19.4	17.8	0.29

FIGURE 7. Benchmarks and measurements.

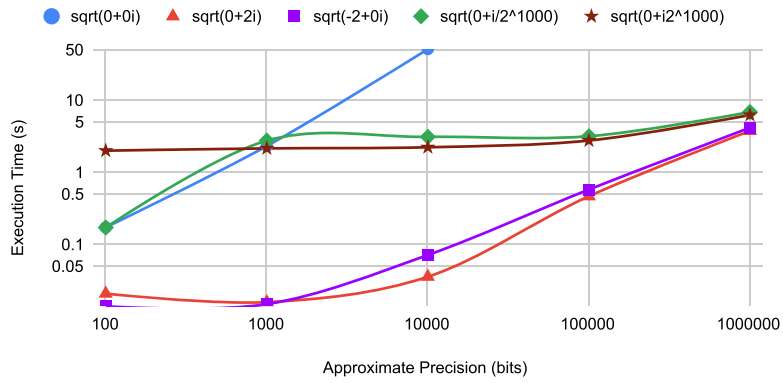


FIGURE 8. Execution time of the extracted complex square root function.

constant overhead associated with scaling the input to the range where Heron method converges. The performance of our complex square root appears to be comparable to the performance of our real square root, which is comparable to a hand-written Haskell/AERN implementation [27].

9 Conclusion and future work

We presented a new axiomatization of reals in a constructive dependent type theory and proved its soundness by extending the standard realizability interpretation from computable analysis. We implemented our theory in Coq and used Coq's code extraction features to generate efficient Haskell programs for exact real computation based on the AERN library.

Our new axiomatization is particularly well-suited for verifying exact real computation programs where real numbers are provided as abstract entities and the underlying representations are hidden to the users. Our axiomatization of reals is representation independent and its realizability interpretation is generically defined so that any implementation of effective real numbers can be used. Hence, though in our example we mapped our axiomatic reals to a specific datatype of real numbers, `AERN2.CReal` in Haskell, it can be mapped also to any other sane datatype of reals. Moreover, by proving that any concrete types satisfying our set of axioms are equivalent, in this sense, we showed that our axiomatization of reals is universal.

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

In future work, we plan to extend our implementation with other important operations on real numbers such as trigonometric and exponential functions and mathematical constants such as π and e and operations like polynomial root finding and matrix diagnonalization, which can be implemented effectively using nondeterministic limits. Further, by extending our framework to real functions, including a sound axiomatization of the continuity principle, e.g. [55], we could also study solution operators for ordinary or partial differential equations by applying recent ideas from real complexity theory [23, 29].

Acknowledgment

The authors thank Franz Braußе and Norbert Müller for helpful discussions and the anonymous reviewers for their careful reading of our manuscript and their insightful comments and suggestions.

Funding

Holger Thies is supported by JSPS KAKENHI Grant numbers JP20K19744 and JP23H03346. Sewon Park is supported by JSPS KAKENHI Grant number JP18H03203 and JSPS KAKENHI (Grant-in-Aid for JSPS Fellows) JP22F22071 as a JSPS International Research Fellow.

This project has received funding from the EU's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 731143.

References

[1] A. Balluchi, A. Casagrande, P. Collins, A. Ferrari, T. Villa and A. L. Sangiovanni-Vincentelli. Ariadne: a framework for reachability analysis of hybrid automata. In *Proceedings of the International Symposium on Mathematical Theory of Networks and Systems*, Kyoto, Japan, 2006.

[2] U. Berger and T. Hou. Coinduction for exact real number computation. *Theory of Computing Systems*, **43**, 394–409, 2008.

[3] U. Berger. Extracting non-deterministic concurrent programs. In *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, J.-M. Talbot and L. Regnier, eds, pp. 26:1–26:21. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2016.

[4] U. Berger and H. Tsuiki. Intuitionistic fixed point logic. *Annals of Pure and Applied Logic*, **172**, 102903, 2021.

[5] L. Birkedal. Developing Theories of Types and Computability via Realizability. *Electronic Notes in Theoretical Computer Science*, **34**, 2, 2020.

[6] E. A. Bishop. *Foundations of Constructive Analysis*, McGraw-Hill, New York, 1967.

[7] S. Boldo, J.-C. Filliâtre and G. Melquiond. Combining Coq and Gappa for certifying floating-point programs. In *Intelligent Computer Mathematics*, J. Carette, L. Dixon, C. S. Coen and S. M. Watt, pp. 59–74. Springer, Berlin, Heidelberg, 2009.

[8] S. Boldo and G. Melquiond. Flocq: A unified library for proving floating-point algorithms in Coq. In *2011 IEEE 20th Symposium on Computer Arithmetic*, Tübingen, Germany, E. Antelo, D. Hough, P. Ienne, eds, pp. 243–252. IEEE Los Alamitos, CA, USA, 2011.

[9] S. Boldo, C. Lelay and G. Melquiond. Formalization of real analysis: a survey of proof assistants and libraries. *Mathematical Structures in Computer Science*, **26**, 1196–1233, 2016.

[10] S. Boldo and G. Melquiond. *Computer Arithmetic and Formal Proofs - Verifying Floating-Point Algorithms with the Coq System*. ISTE Press - Elsevier, London, 2017.

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

[11] S. Boldo, F. Faissole and A. Chapoutot. Round-off error and exceptional behavior analysis of explicit Runge-Kutta methods. *IEEE Transactions on Computers*, **69**, 1745–1756, 2019.
[12] V. Brattka and P. Hertling. Feasible real random access machines. *Journal of Complexity*, **14**, 490–526, 1998.
[13] V. Brattka. The emperor's new recursiveness: the epigraph of the exponential function in two models of computability. In *Words, Languages & Combinatorics III*, M. Ito and T. Imaoka, eds, pp. 63–72. World Scientific Publishing, Singapore, 2003.
[14] F. Braußе, P. Collins and M. Ziegler. Computer science for continuous data. In *Computer Algebra in Scientific Computing*, F. Boulier, M. England, T. M. Sadykov and E. V. Vorozhtsov, eds, pp. 62–82. Springer International Publishing, Cham, 2022.
[15] D. S. Bridges. Constructive mathematics: a foundation for computable analysis. *Theoretical Computer Science*, **219**, 95–109, 1999.
[16] A. Chlipala. *Certified Programming with Dependent Types: A Pragmatic Introduction to the Coq Proof Assistant*. MIT Press, Cambridge, Massachusetts, USA, 2013.
[17] R. L. Cooke. *Classical Algebra: Its Nature, Origins, and Uses*. John Wiley & Sons, Hoboken, New Jersey, USA, 2008.
[18] L. Cruz-Filipe, H. Geuvers and F. Wiedijk. C-CoRN, the constructive Coq repository at Nijmegen. In *International Conference on Mathematical Knowledge Management*, A. Asperti, G. Bancerek, A. Trybulec, eds, pp. 88–103. Springer, Berlin, Heidelberg, 2004.
[19] D. Gallois-Wong, S. Boldo and P. Cuoq. Optimal inverse projection of floating-point addition. *Numerical Algorithms*, **83**, 957–986, 2020.
[20] P. Hertling. A real number structure that is effectively categorical. *Mathematical Logic Quarterly*, **45**, 147–182, 1999.
[21] M. Hofmann. On the interpretation of type theory in locally cartesian closed categories. In L. Pacholski and J. Tiuryn, eds, *Computer Science Logic*, pp. 427–441. Springer, Berlin, Heidelberg, 1995.
[22] B. Jacobs. *Categorical Logic and Type Theory*. Elsevier, Amsterdam, 1999.
[23] A. Kawamura, F. Steinberg and H. Thies. Parameterized complexity for uniform operators on multidimensional analytic functions and ODE solving. In *International Workshop on Logic, Language, Information, and Computation*, L. S. Moss, R. de Queiroz and M. Martinez, eds, pp. 223–236. Springer, Berlin, Heidelberg, 2018.
[24] M. Konečný. Verified exact real limit computation. In *Proceeding of the 15th International Conference on Computability and Complexity in Analysis (CCA)*, pp. 9–10, 2018.
[25] M. Konečný, F. Steinberg and H. Thies. Computable analysis for verified exact real computation. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020)*, N. Saxena and S. Simon, eds, pp. 50:1–50:18. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2020.
[26] M. Konečný. Aern2-real: A Haskell library for exact real number computation. Software available at https://hackage.haskell.org/package/aern2-real.
[27] M. Konečný, S. Park and H. Thies. Axiomatic reals and certified efficient exact real computation. In *International Workshop on Logic, Language, Information, and Computation*, A. Silva, R. Wassermann and R. de Queiroz, eds, pp. 252–268. Springer, Cham, 2021.
[28] M. Konečný, S. Park and H. Thies. Certified computation of nondeterministic limits. In *NASA Formal Methods Symposium*, J.V. Deshmukh, K. Havelund and I. Perez, eds, pp. 771–789. Springer, Cham, 2022.
[29] Ivan Koswara, Svetlana Selivanova and Martin Ziegler. Computational complexity of real powering and improved solving linear differential equations. In *International Computer*

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

Science Symposium in Russia, R. van Bevern and G. Kucherov, eds, pp. 215–227. Springer, Cham, 2019.

[30] R. Krebbers and B. Spitters. Computer certified efficient exact reals in Coq. In *International Conference on Intelligent Computer Mathematics*, J. H. Davenport, W. M. Farmer, J. Urban and F. Rabe, eds, pp. 90–106. Springer, Berlin, Heidelberg, 2011.

[31] C. Kreitz and K. Weihrauch. Theory of representations. *Theoretical Computer Science*, **38**, 35–53, 1985.

[32] J. R. Longley. *Realizability Toposes and Language Semantics*. PhD thesis, University of Edinburgh. College of Science and Engineering, Edinburgh, 1995.

[33] H. Luckhardt. A fundamental effect in computations on real numbers. *Theoretical Computer Science*, **5**, 321–324, 1977.

[34] G. Melquiond. Proving bounds on real-valued functions with computations. In *International Joint Conference on Automated Reasoning*, A. Armando, P. Baumgartner and G. Dowek, eds, pp. 2–17. Springer, Berlin, Heidelberg, 2008.

[35] K. Miyamoto and H. Schwichtenberg. Program extraction in exact real arithmetic. *Mathematical Structures in Computer Science*, **25**, 1692–1704, 2015.

[36] D. Monniaux. The pitfalls of verifying floating-point computations. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, **30**, 1–41, 2008.

[37] N. T. Müller. Implementing limits in an interactive RealRAM. In *3rd Conference on Real Numbers and Computers*, J.-M. Chesneaux, F. Jézéquel, J.-L. Lamotte and J. Vignes, eds, pp. 59–66. Université Pierre et Marie Curie, Paris, 1998.

[38] N. T. Müller. The iRRAM: exact arithmetic in C++. In *International Workshop on Computability and Complexity in Analysis*, J. Blanck, V. Brattka and P. Hertling, eds, pp. 222–252. Springer, Berlin, Heidelberg, 2000.

[39] E. Neumann and A. Pauly. A topological view on algebraic computation models. *Journal of Complexity*, **44**, 1–22, 2018.

[40] R. O'Connor. A monadic, functional implementation of real numbers. *Mathematical Structures in Computer Science*, **17**, 129–159, 2007.

[41] R. O'Connor. Certified exact transcendental real number computation in Coq. In *Theorem proving in higher order logics. Vol. 5170 of Lecture Notes in Computer Science*, pp. 246–261. Springer, Berlin, 2008.

[42] R. O'Connor and B. Spitters. A computer-verified monadic functional implementation of the integral. *Theoretical Computer Science*, **411**, 3386–3402, 2010.

[43] S. Park, F. Braußе, P. Collins, S. Y. Kim, M. Konečný, G. Lee, N. Müller, E. Neumann, N. Preining and M. Ziegler. Semantics, Specification Logic, and Hoare Logic of Exact Real Computation. *Logical Methods in Computer Science*, **20**, 17:1–17:55, 2024.

[44] R. Rettinger, F. Braußе and N. Müller. Intensionality and multi-valued limits. In *Proceeding of the 15th International Conference on Computability and Complexity in Analysis (CCA)*, p. 11, 2018.

[45] B. Reus. Realizability models for type theories. *Electronic Notes in Theoretical Computer Science*, **23**, 128–158, 1999.

[46] T. Ringer, K. Palmskog, I. Sergey, M. Gligoric and Z. Tatlock. QED at large: A survey of engineering of formally verified software. 2020. arXiv:2003.06458.

[47] M. Schröder. Effectivity in spaces with admissible multirepresentations. *Mathematical Logic Quarterly: Mathematical Logic Quarterly*, **48**, 78–90, 2002.

[48] H. Schwichtenberg. Constructive analysis with witnesses. In *Proof Technology and Computation. Nato Science Series*, H. Schwichtenberg and K. Spies, eds, pp. 323–354, IOS Press, Utrecht, 2006.

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025

[49] E. Specker. Nicht konstruktiv beweisbare Sätze der analysis. *The Journal of Symbolic Logic*, **14**, 145–158, 1949.
[50] F. Steinberg, L. Thery and H. Thies. Computable analysis and notions of continuity in Coq. *Logical Methods in Computer Science*, **17**, 16:1–16:43, 2021.
[51] T. Streicher. *Semantics of Type Theory: Correctness, Completeness and Independence Results*. Birkhäuser Boston, MA, 1991.
[52] J. Van Oosten. *Realizability: An Introduction to its Categorical Side*. Elsevier, Amsterdam, 2008.
[53] K. Weihrauch. *Computable Analysis*. Springer, Berlin, 2000.
[54] C. Xu. *A Continuous Computational Interpretation of Type Theories*. PhD thesis, University of Birmingham, Birmingham, 2015.

Received 31 January 2022

Downloaded from https://academic.oup.com/logcom/advance-article/doi/10.1093/logcom/exae066/7824790 by College of Engineering, Nihon University user on 13 February 2025