Certified Computation of Nondeterministic Limits*

Michal Konečný[1][0000−0003−2374−9017], Sewon Park[2][0000−0002−6443−2617], and Holger Thies[3][0000−0003−3959−0741]

[1] Aston University, Birmingham, UK m.konecny@aston.ac.uk
[2] Kyoto University, Kyoto, Japan sewon@kurims.kyoto-u.ac.jp
[3] Kyoto University, Kyoto, Japan thies.holger.5c@kyoto-u.ac.jp

Abstract. The computational content of constructive metric completeness is the operator that computes limits of Cauchy sequences. It can be used to construct certified programs that compute interesting transcendental real numbers from sequences of approximations. The desired nondeterministic version of it would be to nondeterministically compute real numbers from nondeterministic approximations. However, it is not obvious how nondeterministic metric completeness should be formalized. We extend previous work on the formalization of exact real computation by primitive properties of nondeterminism. We show that by these properties, various forms of nondeterministic metric completeness can be derived without extending the axiomatic structure of constructive real numbers. We further implement our theory in the Coq proof assistant and use Coq's code extraction features to extract efficient exact real computation programs using several forms of nondeterministic computation.

Keywords: Constructive Real Numbers · Formal Proofs · Exact Real Number Computation · Program Extraction · Nondeterminism.

1 Introduction

Exact real computation is an elegant approach in which real numbers and other continuous mathematical structures are treated as basic entities in programming languages that can be manipulated exactly without introducing rounding errors. This is often realized by having an abstract data type for real numbers which keeps track of errors in the background and increases the working precision when necessary. Algorithm designers therefore can focus solely on the mathematical problem itself, without thinking about representation issues of real numbers. Of course this elegance comes at a price and exact real arithmetic is usually less efficient than the more common approach of using fixed-length floating point

* Holger Thies is supported by JSPS KAKENHI Grant Number JP20K19744. Sewon Park is supported by JSPS KAKENHI Grant number JP18H03203. This project has received funding from the EU's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 731143. The authors thank Franz Brauße and Norbert Müller for helpful discussions.

approximations. Nonetheless, there are many applications where robustness and reliability are more important than mere efficiency and exact real computation is a feasible alternative in these cases. Furthermore, optimized implementations of exact real computation achieve to minimize the overhead in many cases [21, 1, 16].

In general, the above mentioned properties of exact real computation also facilitate the process of formal verification as it is not necessary to deal with the difficulties of formalizing floating point arithmetic [4]. Indeed, there are already several works dealing with the verification of exact real number computations e.g. [10], [22], [24], [28], [2], and many more and verifying operations like basic arithmetic is usually straightforward.

On the other hand, the continuous semantics of exact real computation come with their own difficulties. In particular, the seemingly simple process of making a decision, that is, choosing one branch of a program if a condition holds and another if it does not, is often non-trivial as it involves discontinuities.

Consider for example the simple comparison operator $<$ on the reals, usually chosen to be a function from reals to the Booleans. As the function is not continuous, there is no way to make this operator computable. More generally, any total, continuous function from the reals to the Booleans is necessarily constant, and thus no interesting operation can be computed.

There are essentially two ways to deal with this problem:

(i) Partiality: Consider the partial function $<$ of type $\mathbb{R} \times \mathbb{R} \rightharpoonup \mathsf{bool}$ such that $x < y$ is undefined if $x = y$. In this case, the semantics are identical to the usual mathematical interpretation, but programs fail to terminate if two equal numbers are compared [30, Theorem 4.1.16].
(ii) Nondeterminism: Extend the notion of computation to multivalued functions $f\colon A \rightrightarrows B$. The comparison may be replaced by a multivalued soft comparison $x <_k y = \{tt \mid x < y + 2^k\} \cup \{ff \mid y < x + 2^k\}$ [19, 6]. That is, if the two numbers are far enough apart, the correct Boolean value will be returned, but if the numbers are close, any of the two values can be returned nondeterministically.

While (i) is a simple solution, non-termination of programs is usually extremely undesirable. Exact real computation software therefore often implements primitive operations to construct nondeterministic functions. Examples include AERN's `select` [16], or `choose` in iRRAM [21] and Ariadne [1, 8]. These frameworks are used to compute highly accurate approximations of numerical problems and nondeterministic operations have been applied in practical situations. However, the formal semantics of this kind of nondeterminism has been less studied.

In recent work [14], we presented a formalization of constructive real numbers in a simple dependent type theory. Our formalization was designed as a framework to extract certified exact real computation programs from constructive proofs, and closely model some of the features of exact real computation such as nondeterminism. For example, when proving a theorem of the form $\Pi(x : \mathsf{R}).\ P\,x \to \mathsf{M}\Sigma(y : \mathsf{R}).\ Q\,x\,y$, 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. This is realized by mapping the axiomatized real number type R in the type theory to the abstract data type of real numbers in an exact real computation software. In the cited paper, the practicality of the approach is demonstrated by implementing the axiomatization in Coq and mapping to data types and operators in AERN using Coq's program extraction mechanism [17, 18]. A main goal of our implementation is to not only provide provably correct but also efficient computation, comparable to native implementations in exact real computation frameworks.

A unique feature 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 [5, 23]. While it is clear how the limit computation should be axiomatized in the logical language [27, 3, 7], it has been under debate how to deal with the case when nondeterminism is involved in the limit computation [11, 15]. Note that this situation occurs quite naturally even for simple operations such as computing square roots of complex numbers.

Early versions of iRRAM therefore already provided a simple nondeterministic limit operation as primitive [21, § 10.3] which the authors of the software recently suggested to replace by a more generic operation for nondeterministic limits [11]. However, an important open question that remained is, besides the practicality, if the nondeterministic limit operation is sound, natural, and primitive, i.e. needs to be introduced as an axiom in the axiomatization of constructive real numbers or if it can be derived from a more general principle.

Inspired by [11] and personal communication with the authors, we were able to answer this question: In the current work we specify *nondeterministic dependent choice*, a simple and natural principle of the nondeterminism itself that makes the limit operation suggested in [11] and some other forms of nondeterministic limits derivable. It automatically ensures that the nondeterministic limit operations are sound and that they naturally arise due to the characteristics of nondeterminism applied to the ordinary metric completeness. It moreover suggests that, assuming the computational language is rich enough, there is no need to introduce a nondeterministic limit primitive in exact real computation.

To demonstrate the practicality, we extended our Coq implentation proposed in [14] in the suggested way. Our implementation offers a framework where users can obtain certified programs using nondeterministic limits simulated in the AERN framework. We present some basic examples, extract AERN programs from them and show that they behave well in terms of efficiency.

2 Background and Overview

Let us first briefly summarize the theory from [14] and describe some minor modifications to the original work. Although we mostly have a concrete implementation in the Coq proof assistant in mind, we formulate our results in a more general type-theoretic setting and hope that this also makes it accessible

to readers less familiar with Coq. Formal descriptions of dependent type theories can be found in various literature including [29, Chapter 1].

We assume to work in a dependent type theory with 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 (with an implicit type level). We assume that the identity types $=$ are in Prop and that Prop is a type universe closed under $\to, \times, \vee, \exists, \Pi$, containing two types $\mathsf{True}, \mathsf{False} : \mathsf{Prop}$ which are the unit and the empty type respectively. It is a universe of classical propositions in that for example $P \vee Q : \mathsf{Prop}$ denotes the classical fact that P or Q holds which differs from $P + Q : \mathsf{Type}$ the sum type denoting there to be a computational procedure deciding if P or Q holds. Similarly, when we have a family of classical propositions $P : X \to \mathsf{Prop}$, the type $\exists(x : X).\ P\,x : \mathsf{Prop}$ belonging to Prop denotes the classical existence of $x : X$ satisfying $P\,x$ while the ordinary dependent pair type (also called Σ-type), $\Sigma(x : X).\ P\,x : \mathsf{Type}$ belonging to Type denotes the constructive existence. Note that Prop shares the same constructs for function types (implications) $\to$, product types (conjunctions) $\times$, and dependent function types, also called Π-types, (universal quantifiers) Π with Type.

In order to make Prop classical, we assume the (classical) law of excluded middle $\Pi(P : \mathsf{Prop}).\ P \vee \neg P$ (where $\neg P :\equiv P \to \mathsf{False}$), the (classical) propositional extensionality $\Pi(P, Q : \mathsf{Prop}).\ (P \leftrightarrow Q) \to P = Q$ (where $P \leftrightarrow Q :\equiv (P \to Q) \times (Q \to P)$), and the (classical) countable choice of the form $\Pi(A : \mathsf{Type}).\ \Pi(P : \mathsf{N} \to A \to \mathsf{Prop}).\ (\Pi(n : \mathsf{N}).\ \exists(x : X).\ P\,n\,x) \to \exists(f : \mathsf{N} \to A).\ \Pi(n : \mathsf{N}).\ P\,n\,(f\,n)$. Note the use of the classical $\exists$, as opposed to the constructive Σ and that we did not assume countable choice in [14].

We also assume the general functional extensionality $\Pi(A : \mathsf{Type}).\ \Pi(P : A \to \mathsf{Type}).\ \Pi(f, g : \Pi(x : A).\ P\,x).\ \big(\Pi(x : A).\ f\,x = g\,x\big) \to f = g$ and the Markov principle $\Pi(f : \mathsf{N} \to \mathsf{Prop}).\ \big(\Pi(n : \mathsf{N}).\ (f\,n) + \neg(f\,n)\big) \to (\exists(n : \mathsf{N}).\ f\,n) \to \Sigma(n : \mathsf{N}).\ f\,n$.

From [14], we keep the axiomatization of *Kleenean*, saying that there is a type constant K admitting two distinct constants $\mathsf{true}, \mathsf{false} : \mathsf{K}$. We write $\lceil k \rceil$ for $k = \mathsf{true}$. K denotes semi-decidable decision procedures. For example, we assume $\Pi(x, y : \mathsf{R}).\ \Sigma(k : \mathsf{K}).\ \lceil k \rceil = (x < y)$ to say that comparison over the reals is semi-decidable. Here $<$ of type $\mathsf{R} \to \mathsf{R} \to \mathsf{Prop}$ is an axiomatized term constant.

We keep all the axioms of real numbers from op. cit. except for classical completeness. In [14] we had two different formulations of completeness. The first is constructive completeness saying that for any $f : \mathsf{N} \to \mathsf{R}$ which is a (fast) Cauchy sequence, there constructively is a real number $x : \mathsf{R}$ which is the limit point of the sequence. The other is classical completeness which says for any classical predicate $P : \mathsf{R} \to \mathsf{Prop}$ that is classically nonempty and bounded above, there classically exists the least upper bound. In our modified theory we can prove that constructive completeness implies classical completeness using classical countable choice and hence removed the classical completeness axiom.

The soundness of the axioms in [14] is argued by extending a realizability interpretation in the category of assemblies over Kleene's second algebra [13, 26] by mapping types into assemblies and terms into morhpisms in the category [25,

§ 4 and § 5]. An assembly over Kleene's second algebra is a pair of a set A and a binary relation $\Vdash_A \subseteq \mathbb{N}^{\mathbb{N}} \times A$ that is surjective in the sense that for any $x \in A$, there is $\varphi \in \mathbb{N}^{\mathbb{N}}$ such that $(\varphi, x) \in \Vdash_A$. The binary relation is often written in infix notation and φ is said to realize x when $\varphi \Vdash_A x$ holds. Given two assemblies $(A, \Vdash_A)$ and $(B, \Vdash_B)$, a function $f : A \to B$ is defined continuous (computable) if there is a continuous (computable[4]) partial Baire space function $\tau : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ that tracks f in the sense that for any $(\varphi, x) \in \Vdash_A$, it holds that $\tau(\varphi) \Vdash_B f(x)$. The category of assemblies over Kleene's second algebra is the category of such assemblies and computable functions.

In [14], we propose a nondeterminism monad M such that for any type $X : \mathsf{Type}$, we automatically get a type $\mathsf{M}X : \mathsf{Type}$ modeling the result of a nondeterministic computation in X. Formally speaking, we consider a type transformer $F : \mathsf{Type} \to \mathsf{Type}$ a monad in our type theory when it is accompanied with

(i) a function lift $\mathsf{lift}^F : \Pi(X, Y : \mathsf{Type}).\ (X \to Y) \to (F\,X) \to F\,Y$ (we write $\mathsf{lift}^F_{X,Y}$ for $\mathsf{lift}^F\,X\,Y$),
(ii) a proof that F and lift^F form a functor

$$\mathsf{lift}^F_{X,X}(\mathrm{id}_X) = \mathrm{id}_{F\,X} \text{ where } \mathrm{id}_{X:\mathsf{Type}} :\equiv \lambda(x : X).\ x, \text{ and}$$

$$\mathsf{lift}^F_{X,Z}(f \circ g) = (\mathsf{lift}^F_{Y,Z} f) \circ (\mathsf{lift}^F_{X,Y} g) \text{ where } f \circ g :\equiv \lambda(x : X).\ f\,(g\,x)$$

for all $X, Y, Z : \mathsf{Type}$, $g : X \to Y$, and $f : Y \to Z$,
(iii) a unit $\mathsf{unit}^F : \Pi(X : \mathsf{Type}).\ X \to F\,X$ (we write unit^F_X for $\mathsf{unit}^F\,X$),
(iv) a proof that the unit is a natural transformation

$$(\mathsf{lift}^F_{X,Y} f) \circ \mathsf{unit}^F_X = \mathsf{unit}^F_Y \circ f$$

for all $X, Y : \mathsf{Type}$ and $f : X \to Y$,
(v) a multiplication $\mathsf{mult}^F : \Pi(X : \mathsf{Type}).\ (F\,(F\,X)) \to F\,X$ (we write mult^F_X for $\mathsf{mult}^F\,X$),
(vi) a proof that the multiplication is a natural transformation

$$\mathsf{mult}^F_Y \circ (\mathsf{lift}^F_{F\,X,F\,Y}(\mathsf{lift}^F_{X,Y} f)) = (\mathsf{lift}^F_{X,Y} f) \circ \mathsf{mult}^F_X$$

for all $X, Y : \mathsf{Type}$ and $f : X \to Y$, and

(vii) proofs of the three monad coherence conditions

- $\mathsf{mult}^F_X \circ \mathsf{unit}^F_{F\,X} = \mathrm{id}_{F\,X}$
- $\mathsf{mult}^F_X \circ (\mathsf{lift}^F_{X,F\,X} \mathsf{unit}^F_X) = \mathrm{id}_{F\,X}$
- $\mathsf{mult}^F_X \circ \mathsf{mult}^F_{F\,X} = \mathsf{mult}^F_X \circ (\mathsf{lift}^F_{F\,(F\,X),F\,X} \mathsf{mult}^F_X)$

for all $X : \mathsf{Type}$. Note the analogy in the definition with monads in category theory.

The monad M in the type theory is interpreted as a monad $\mathbf{M}$ in the category of assemblies over Kleene's second algebra whose action on an assembly $(A, \Vdash_A)$

[4] in the sense of computable analysis [30]

is $(\mathbf{P}_+(A), \Vdash_{\mathbf{M}A})$ where $\mathbf{P}_+(A)$ is the set of nonempty subsets of A and the realization relation is defined by

$$\varphi \Vdash_{\mathbf{M}A} S \quad :\Leftrightarrow \quad \exists x \in A.\, \varphi \Vdash_A x.$$

The monad on a function is defined by $\mathbf{M}(f) := S \mapsto \bigcup_{x \in S} \{f(x)\}$. Note that the lifted function is tracked by the same Baire space function that tracks f. The monad is interesting as it classifies computable nondeterministic functions in computable analysis.[5]

The main novel contributions in this work deal with extending the formalization of nondeteterminism from our previous work. We suggest a different set of axioms for the nondeterminism monad. Our new formalization is expressive enough to define various notions of nondeterministic completeness useful in practical applications. The new set of axioms is more expressive in the sense that all the properties of nondeterminism used in [14] are still derivable.

3 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 a nondeterministic version $\mathsf{M}X : \mathsf{Type}$ of it. A term of the nondeterministic type is regarded as the result of a nondeterministic computation in X. As the underlying set of $\mathsf{M}X$ in the model is the set of non-empty subsets of the underlying set of X, we suggest a characterization of the monad by relating it with the classical non-empty 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 it forms a monad with function lift, unit, and multiplication:

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

Here, the occurrences of $-$ represent some classical proof terms. A (dependent) pair (a, b) such that $a : X$ and $b : P\,a$ is of type $\Sigma(x : X).\ P\,x$ or $\exists(x : X).\ P\,x$ where the ambiguity is only for the simplicity in our presentation. And, π_1 is the first projection of pairs (Σ-types).

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 $\mathsf{mult}^{\mathsf{M}}$ which form a monad in our type theory. In order to relate it with the classical non-empty power-set monad, we assume that there is a submonoidal natural transformation

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

[5] Nondeterministic functions are also known as *multivalued* functions.

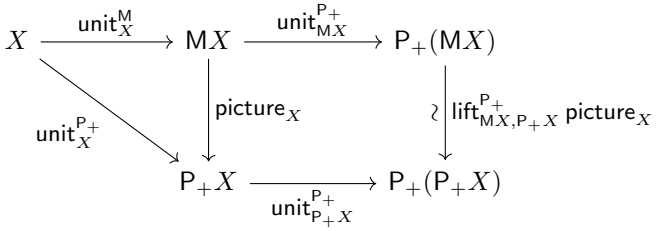
which we write $\mathsf{picture}_X$ for $\mathsf{picture}\,X$. It is a submonoidal natural transformation in that (i) it is a natural transformation, (ii) for any $X : \mathsf{Type}$, $\mathsf{picture}_X$ is monic $\Pi(x, y : \mathsf{M}X).\ \mathsf{picture}_X\, x = \mathsf{picture}_X\, y \to x = y$, and (iii) the coherence conditions which on the unit is $\mathsf{picture}_X \circ \mathsf{unit}^{\mathsf{M}}_X = \mathsf{unit}^{\mathsf{P}_+}_X$ and on the multiplication is $\mathsf{picture}_X \circ \mathsf{mult}^{\mathsf{M}}_X = \mathsf{mult}^{\mathsf{P}_+}_X \circ \mathsf{picture}_{\mathsf{P}_+X} \circ (\mathsf{lift}^{\mathsf{M}}_{\mathsf{M}X\,\mathsf{P}_+X}\mathsf{picture}_X)$ hold.

We call the natural transformation "picture" because we regard $\mathsf{picture}_X\, x$, when $x : \mathsf{M}X$ is a nondeterministic element, as a classical picture showing the elements x represents. Let us make the definition

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

which discards the second entry of $\mathsf{picture}_X\, x$ such that we can conveniently use $\mathsf{pic}_X\ x\ y : \mathsf{Prop}$ to express that $y : X$ is a possible outcome of $x : \mathsf{M}X$.

We further characterize the nondeterminism by that the classically lifted picture $\mathsf{lift}^{\mathsf{P}_+}_{\mathsf{M}X,\mathsf{P}_+X}\,\mathsf{picture}_X : \mathsf{P}_+(\mathsf{M}X) \to \mathsf{P}_+(\mathsf{P}_+X)$ constructively admits an inverse; i.e., $\Pi(X : \mathsf{Type}).\ \Sigma(i : \mathsf{P}_+(\mathsf{P}_+X) \to \mathsf{P}_+(\mathsf{M}X)).\ \big(i \circ (\mathsf{lift}^{\mathsf{P}_+}_{\mathsf{M}X,\mathsf{P}_+X}\,\mathsf{picture}_X) = \mathrm{id}_{\mathsf{P}_+(\mathsf{M}X)}\big) \times \big((\mathsf{lift}^{\mathsf{P}_+}_{\mathsf{M}X,\mathsf{P}_+X}\,\mathsf{picture}_X) \circ i = \mathrm{id}_{\mathsf{P}_+(\mathsf{P}_+X)}\big)$ holds. The following diagram shows the relation between the two monads.



The last building block in relating the two monads is a destruction method. When we have a nondeterministic object $x : \mathsf{M}X$, we assume that we can obtain a term of type $\mathsf{M}\Sigma(y : X).\ \mathsf{pic}_X\, x\, y$. 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.

We carry some of the original characterizations of the nondeterministic monad from [14]. 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:

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

We further carry over the assumption that if a type X is subsingleton, we can eliminate the nondeterminism on $\mathsf{M}X$:

$$\mathsf{elimM} : \Pi(X : \mathsf{Type}).\ (\Pi(x, y : X).\ x = y) \to (\mathsf{M}\ X) \to X.$$

Remark 1. When we have a nondeterministic object $x : \mathsf{M}X$, regarding it as the result of some nondeterministic computation, it is desirable to analyze properties of the possible outcomes of the nondeterministic computation x. For a classical predicate $P : X \to \mathsf{Prop}$, we can express *all possible outcomes* $y : X$ *of* x *satisfy* $P\,y$ by $\Pi^{\mathsf{M}}(y : x).\ P\,y :\equiv \Pi(y : X).\ \mathsf{pic}_X\, x\, y \to P\,y$ and *some possible outcomes* $y : X$ *of* x *satisfy* $P\,y$ by $\exists^{\mathsf{M}}(y : x).\ P\,y :\equiv \exists(y : X).\ (\mathsf{pic}_X\, x\, y) \times (P\,y)$.

3.1 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, in the computational point of view, when we apply it repeatedly, we get, nondeterministically, 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.

In our type theory, we already have a tool to express repeated applications, the primitive recursion $\mathsf{N\text{-}rec}_{\lambda(n:\mathsf{N}).\ \mathsf{M}\,(P\,n)}$ which 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 $P\,n :\equiv \mathsf{R}$ 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 $1/2$, we expect to have one of the two sequences $1/2, 0, 0, 0, 0, \cdots$ or $1/2, 1, 1, 1, 1, \cdots$ nondeterministically. However, when we apply the primitive recursion, all we can get is the sequence of the nondeterministic real numbers $1/2, (0 \text{ or } 1), (0 \text{ or } 1), \cdots$ 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 behavior identical to primitive recursion.

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

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

an M-*trace* of Q. Note that admitting a trace automatically ensures that Q is a (classically) entire relation: $\Pi(n : \mathsf{N}).\ \Pi(x : P\,n).\ \exists(y : P\,(n+1)).\ Q\,n\,x\,y$.

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

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

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

Given any M-trace f of Q, 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 which is shown in the beginning of this subsection. The second is to apply the following 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 (c.f. Figure 1).

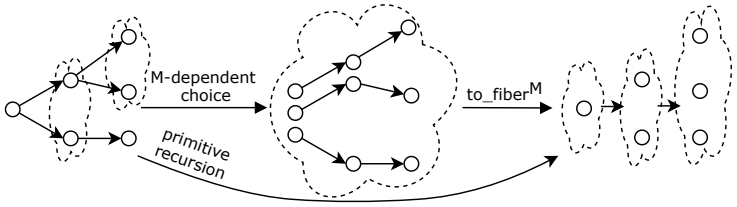


Fig. 1. Intuitive picture on the coherence condition for the M-dependent choice

We assume that our type theory admits M-dependent choice.

Remark 2. The name nondeterministic dependent choice comes from the observation that when repeating the above 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.

Remark 3. In [14], it was 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, when we have a sequence of nondeterministic computations, we can nondeterministically choose one sequence of (deterministic) computations.

The property is used to derive an operator computing deterministic limits from nondeterministic sequences. Observe that the $\omega\mathsf{lift}$ can be derived from the M-dependent choice when we simply let $Q\,n\,x\,y :\equiv \mathsf{True}$. Hence, we conclude that the new set of axioms for the nondeterminism presented in this paper is more expressive than the previous one.

4 Nondeterministic Limits

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 converge to a deterministic point, (2) when a sequence of nondeterministic real numbers converge to a deterministic point, and (3) when a sequence of nondeterministic real numbers converge to a nondeterministic point. The first case is exactly the ordinary metric completeness which is realized by the primitive limit operations in exact real number computation software.

In this section, we derive the other forms of limits from the ordinary constructive completeness and our new more expressive nondeterminism monad.

4.1 Deterministic Limits of Nondeterministic Sequences

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 $f : \mathsf{N} \to \mathsf{MR}$ that is a (fast) Cauchy sequence meaning that

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

(see Remark 1 for the definition of Π^{M}). That is, any choices of $f\ n$ and $f\ m$ will be at most $2^{-(n+m)}$ far apart from each other and thus any possible sequence will be a fast Cauchy sequence, converging to a unique limit point that we define by the relation

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

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

Lemma 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.

In practice, 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:

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

Here, $\exists!(x : X).\ P\,x$ is for the classical unique existence abbreviating $\exists(x : X).\ (P\,x) \times \Pi(y : X).\ P\,x \to x = y$.

Example 1 (Real square root). Any non-negative real number classically admits a unique non-negative square root. Consider any real number $x : \mathsf{R}$ with a term of type $x \geq 0$. Let $S :\equiv \lambda(y : \mathsf{R}).\ (y \geq 0) \times (x = y \times y)$. Of course, the classical property can be proven in our system which will yield a term of type $\exists!(y : \mathsf{R}).\ S\,y$. In [14], we use Heron's method with nondeterministic scaling to nondeterministically approximate the non-negative square root. Heron's method is a simple and well known method to approximate the square root of a real number x by the inductively defined sequence $x_0 := 1$ and $x_{i+1} := \frac{1}{2}\left(x_i + \frac{x}{x_i}\right)$. The sequence converges quadratically to $\sqrt{x}$ in the interval $\frac{1}{4} \leq x \leq 2$, meaning $|\sqrt{x} - x_i| \leq 2^{-2^i}$ and thus can be used to construct a fast Cauchy sequence converging to the square root for any x in said interval. Outside of this interval, we can nondeterministically find a scaling factor $z : \mathsf{Z}$ such that $\frac{1}{4} \leq 4^z x \leq 2$, approximate the square root of the scaled number and rescale it appropriately.

Applying the second version of the metric completeness to the defined sequence, we can obtain

$$\surd : \Pi(x : \mathsf{R}).\ x \geq 0 \to \Sigma(y : \mathsf{R}).\ (y \geq 0) \times (x = y \times y).$$

Note that we need to consider the case $x = 0$ separately, see [14] for details.

4.2 Nondeterministic Limits

Suppose we are given a classical description of real numbers $S : \mathsf{R} \to \mathsf{Prop}$ that is classically sequentially closed. Define $\mathsf{is_seq_closed}\,S$ to for 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 (possibly different) real number in S which is at most 2^{-n-1} apart from x_n. That is, a nondeterministic refinement procedure is 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 will show that given such a nondeterministic refinement procedure, we can apply the M-dependent choice to nondeterministically get a point in S which we call the limit point of the procedure. To this end, we define

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

See that the refinement procedure f can be easily adjusted to become a M-trace of Q. The M-dependent choice on it 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. $\Pi(n : \mathsf{N}).\ |(g\,n) - (g\,(n+1))| \leq 2^{-n-1}$, and converges to S's elements, 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 the $\mathsf{lift}^{\mathsf{M}}$ on the procedure and postcomposing it to the result of the M-dependent choice yields the nondeterministic limit.

Theorem 1. *Within our type theory, we can construct a term of type*

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

4.3 Nondeterministic Limits with Additional Information

The nondeterministic refinement in the previous subsection requires to consecutively refine any possible previous approximation to a better approximation of a limit. However, in practice it is often more reasonable to think of the case where all possible approximations throughout the indefinite refinement procedure share some invariant properties. That is, we only have to consider a subset of all possible 2^{-n} approximations to be given as inputs of the n'th refinement.

Let us, for example, again consider the nondeterministic function

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

from Section 3.1. 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 an admissible 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 applying f, we only encounter either 0 or 1 when $n > 0$.

We would like to use the invariant property of f 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 $I : \mathsf{N} \to \mathsf{R} \to \mathsf{Type}$ that is preserved throughout the refinements. We can encode I in P at the step of applying the M-dependent choice:

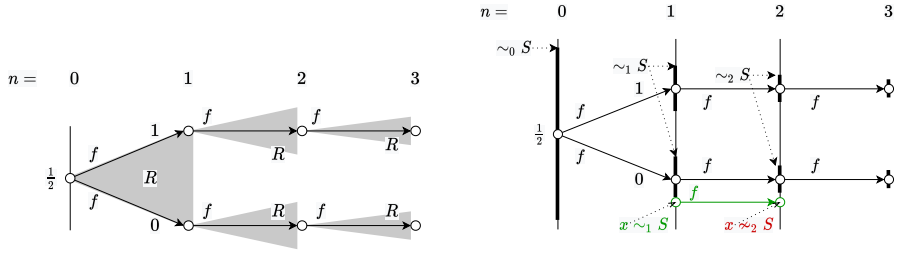
$$
P\,n :\equiv \Sigma(x : \mathsf{R}).\ (x \sim_n S) \times I\,n\,x.
$$

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

Theorem 2. *Within our type theory, we can construct a term of type*

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

Note that the required nondeterministic refinement procedure accepts additional information on its input $I\,n\,x : \mathsf{Type}$, 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 $I\,n\,x :\equiv n > 0 \to (x = 0) + (x = 1)$ such that in 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 2 illustrates this example.



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

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

Fig. 2. Using an invariant property of f to define a limit

Remark 4. 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 [20].

5 Examples

Let us illustrate the use of the limit operations introduced in the previous section with some examples. Multivalued functions play an important role in complex analysis and the area therefore provides a multitude of examples including n-th roots, logarithms and inverse trigonometric functions. In Section 5.1, we present what is perhaps the simplest of these examples, the complex square root.

A second, more theoretical, example where nondeterministic approximation turns out to be useful, is to show that any two real number types that satisfy our axioms are isomorphic. We present this example in Section 5.2.

5.1 Computing Complex Square Roots

We prove the constructive and nondeterministic existence of square roots of complex numbers using a simple method described e.g. in [20]. 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 square roots. Thus, for every $z \neq 0$ there are exactly two square roots. In the case of the square root on nonnegative reals (Example 1) 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.

In order to express the statement, we first extend our theory to a type of complex numbers. We define the type as a pair of real numbers, i.e. $\mathsf{C} :\equiv \mathsf{R} \times \mathsf{R}$ with its field operations and a maximum norm $|\cdot| : \mathsf{C} \to \mathsf{R}$.

For any sequence $f : \mathsf{N} \to \mathsf{C}$ that is Cauchy, we can untangle the real and imaginary parts and obtain the limit point by applying the ordinary limit operator. Similarly, we can extend the nondeterministic limit operator to C.

Let us now return to the square root operation. 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. [9, §6]).

Let $z = a + ib$, then

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

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 $\mathrm{sgn}(a)$ instead of $\mathrm{sgn}(b)$ is used).

Thus, using Example 1, we can show the following restricted version of the existence of a complex square root

$$\sqrt{}_0 : \Pi(z : \mathsf{C}).\ z \neq 0 \to \mathsf{M}(\Sigma(x : \mathsf{C}).\ x \cdot x = z). \tag{1}$$

Finally, we apply Theorem 2 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 I 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 I. 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 (1) 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 n-th 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 $I\, n\, x : \mathsf{Type}$ defined by

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

Applying Theorem 2 with this I, we get

$$\sqrt{} : \Pi(z : \mathsf{C}).\ \mathsf{M}(\Sigma(x : \mathsf{C}).\ x \cdot x = z).$$

5.2 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 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 [12] 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 [30, Theorem 4.3.1]. Then, by scaling, we can construct a term of type

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

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 $\mathsf{is_limit}\, x\, (\lambda(n : \mathsf{N}).\ (g\, n) \cdot 2^{-n})$ 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 (Lemma 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 analyze 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.

6 Implementation and Experimental Results

All mathematical concepts and results presented in this paper have been fully formalized in Coq. The formalization is released under the MIT open-source licence and is included in https://github.com/holgerthies/coq-aern.

Moreover, we extracted Haskell/AERN code from our formalisation of the complex square root that uses our nondeterministic limit operator. The extraction mechanism realizes our axioms as appropriate Haskell/AERN terms, including axioms for real number operations and the nondeterministic choice operator as described in [14, Appendix B] adapted for the changes described here. In particular, M-dependent choice translates to ordinary primitive recursion in the Haskell translation. This is due to the fact that M itself vanishes, i.e., it becomes the Haskell's identity monad, as real number computations in Haskell/AERN are intrinsically nondeterministic already.

Fig. 3 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 constant overhead associated with scaling the input to the range where Heron method converges. In [14] we evaluated the performance of our implementation of the real square root and showed that it is comparable to a hand-written Haskell/AERN implementation. Comparing with the execution times from Fig. 3, it can be seen that the performance of our complex square root appears to be comparable to the performance of the real square root and thus is in turn again comparable to a hand-written implementation.

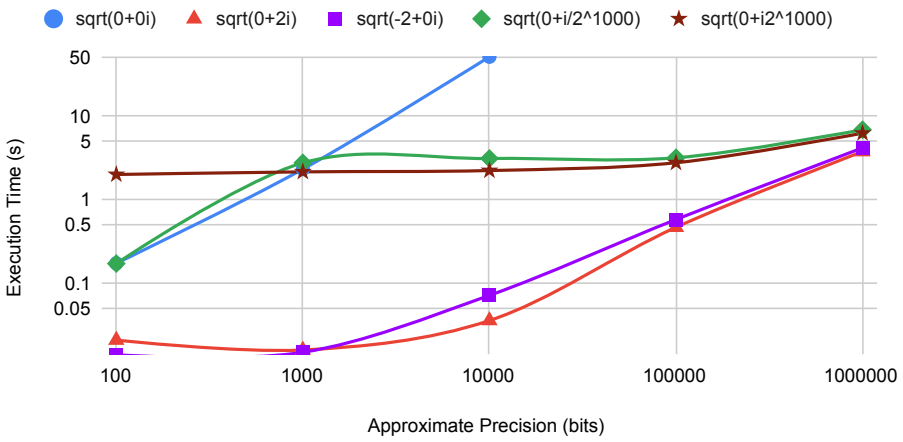


Fig. 3. Execution time of the extracted complex square root function

7 Conclusion

Nondeterminism and the computation of limits are two central features of exact real computation. Extending our previous work, in the current work we devised a sound and powerful framework for formal verification of exact real computation which allows the use of nondeterministic computation in multiple ways. Following recent discussions on the implementation of multivalued limits in exact real computation frameworks, we concluded that we do not need to include a multivalued limit as a primitive operation, but can derive several useful forms of multivalued limit operations from a more natural principle in our theory.

As a simple but important example for a function where nondeterministic limits turn out to be useful, we proved the existence of a complex square root in our constructive theory. Of course, there are several other examples of functions that are necessarily multivalued and we plan to add some of these in future work. We further plan to extend our framework by polynomial root finding and matrix diagnonalization which essentially require nondeterministic limits.

References

1. Balluchi, A., Casagrande, A., Collins, P., Ferrari, A., Villa, T., Sangiovanni-Vincentelli, A.: Ariadne: a Framework for Reachability Analysis of Hybrid Automata. In: Proc. 17th Int. Symp. on Mathematical Theory of Networks and Systems. Kyoto (2006)
2. Berger, U., Tsuiki, H.: Intuitionistic fixed point logic. Ann. Pure Appl. Log. **172**(3), 102903 (2021). https://doi.org/10.1016/j.apal.2020.102903, https://doi.org/10.1016/j.apal.2020.102903
3. Bishop, E.A.: Foundations of constructive analysis (1967)

4. Boldo, S., Melquiond, G.: Computer Arithmetic and Formal Proofs - Verifying Floating-point Algorithms with the Coq System. ISTE Press (2017), https://www.elsevier.com/books/computer-arithmetic-and-formal-proofs/boldo/978-1-78548-112-3
5. Brattka, V.: The emperor's new recursiveness: The epigraph of the exponential function in two models of computability. In: Ito, M., Imaoka, T. (eds.) Words, Languages & Combinatorics III. pp. 63–72. World Scientific Publishing, Singapore (2003), iCWLC 2000, Kyoto, Japan, March 14–18, 2000
6. Brattka, V., Hertling, P.: Feasible real random access machines. Journal of Complexity **14**(4), 490–526 (1998). https://doi.org/https://doi.org/10.1006/jcom.1998.0488, https://www.sciencedirect.com/science/article/pii/S0885064X98904885
7. Bridges, D.S.: Constructive mathematics: a foundation for computable analysis. Theoretical Computer Science **219**(1), 95–109 (1999). https://doi.org/https://doi.org/10.1016/S0304-3975(98)00285-0, https://www.sciencedirect.com/science/article/pii/S0304397598002850
8. Collins, P., Geretti, L., Casagrande, A., Zapreev, I., Zivanovic, S.: Ariadne. http://www.ariadne-cps.org/ (2005-20)
9. Cooke, R.L.: Classical algebra: its nature, origins, and uses. John Wiley & Sons (2008)
10. Cruz-Filipe, L., Geuvers, H., Wiedijk, F.: C-CoRN, the constructive Coq repository at Nijmegen. In: International Conference on Mathematical Knowledge Management. pp. 88–103. Springer (2004)
11. Franz Brausse, Norbert Müller, R.R.: Intensionality and Multi-Valued Limits. In: Proc. 15th Internat. Conf. on Computability and Complexity in Analysis (CCA). p. 11 (2018)
12. Hertling, P.: A real number structure that is effectively categorical. Math. Log. Q. **45**, 147–182 (1999). https://doi.org/10.1002/malq.19990450202, https://doi.org/10.1002/malq.19990450202
13. Hofmann, M.: On the interpretation of type theory in locally cartesian closed categories. In: Pacholski, L., Tiuryn, J. (eds.) Computer Science Logic. pp. 427–441. Springer Berlin Heidelberg, Berlin, Heidelberg (1995)
14. Konečný, M., Park, S., Thies, H.: Axiomatic reals and certified efficient exact real computation. In: International Workshop on Logic, Language, Information, and Computation. pp. 252–268. Springer (2021)
15. Konečný, M.: Verified Exact Real Limit Computation. In: Proc. 15th Internat. Conf. on Computability and Complexity in Analysis (CCA). pp. 9–10 (2018)
16. Konečný, M.: aern2-real: A Haskell library for exact real number computation. https://hackage.haskell.org/package/aern2-real (2021)
17. Letouzey, P.: A new extraction for coq. In: International Workshop on Types for Proofs and Programs. pp. 200–219. Springer (2002)
18. Letouzey, P.: Extraction in coq: An overview. In: Conference on Computability in Europe. pp. 359–369. Springer (2008)
19. Luckhardt, H.: A fundamental effect in computations on real numbers. Theoretical Computer Science **5**(3), 321 – 324 (1977). https://doi.org/https://doi.org/10.1016/0304-3975(77)90048-2, http://www.sciencedirect.com/science/article/pii/0304397577900482
20. Müller, N.T.: Implementing limits in an interactive realram. In: 3rd Conference on Real Numbers and Computers, 1998, Paris. vol. 13, p. 26 (1998)
21. Müller, N.T.: The iRRAM: Exact arithmetic in C++. In: International Workshop on Computability and Complexity in Analysis. pp. 222–252. Springer (2000)

22. Müller, N.T., Uhrhan, C.: Some steps into verification of exact real arithmetic. In: NASA Formal Methods Symposium. pp. 168–173. Springer (2012)
23. Neumann, E., Pauly, A.: A topological view on algebraic computation models. Journal of Complexity **44**, 1–22 (2018)
24. Park, S., Braußе, F., Collins, P., Kim, S., Konečný, M., Lee, G., Müller, N., Neumann, E., Preining, N., Ziegler, M.: Foundation of computer (algebra) analysis systems: Semantics, logic, programming, verification. arXiv e-prints pp. arXiv–1608 (2016)
25. Reus, B.: Realizability models for type theories. Electronic Notes in Theoretical Computer Science **23**(1), 128–158 (1999)
26. Seely, R.A.G.: Locally cartesian closed categories and type theory. Mathematical Proceedings of the Cambridge Philosophical Society **95**(1), 33–48 (1984). https://doi.org/10.1017/S0305004100061284
27. Specker, E.: Nicht konstruktiv beweisbare Sätze der Analysis. The Journal of Symbolic Logic **14**(3), 145–158 (1949)
28. Steinberg, F., Thery, L., Thies, H.: Computable analysis and notions of continuity in Coq. Logical Methods in Computer Science **Volume 17, Issue 2** (May 2021), https://lmcs.episciences.org/7478
29. Univalent Foundations Program, T.: Homotopy Type Theory: Univalent Foundations of Mathematics. https://homotopytypetheory.org/book, Institute for Advanced Study (2013)
30. Weihrauch, K.: Computable analysis. Springer, Berlin (2000)