An axiomatization of real numbers in constructive type theory and its application

Sewon Park (Kyoto University)
j.w.w. Michal Konečný (Aston U) and Holger Thies (Kyoto U)

Feb 12, 2022

2022 China-South Korea Non-Classical Logics and Related Algebra Systems Academic Seminar
Online

Motivation and Overview

- certified computations over real numbers from constructive proofs
- axiomatize real number computation in constructive type theory
- properties of real number computation have to be reflected in the axiomatization

  partiality and nondeterminism
- propose a sound axiomatization of partial and nondeterministic computation, and real numbers
- implement the axiomatization

Motivation and Overview

- certified computations over real numbers from constructive proofs
- axiomatize real number computation in constructive type theory
- properties of real number computation have to be reflected in the axiomatization

  partiality and nondeterminism
- propose a sound axiomatization of partial and nondeterministic computation, and real numbers
- implement the axiomatization

Talk overview:

1. Intro: constructive type theory + why axiomatization of reals not trivial
2. Axiomatization of partiality, nondeterminism, and reals
3. Some examples
4. Implementation and evaluation results
5. Conclusion

1. **Intro: constructive type theory + why axiomatization of reals not trivial**
2. Axiomatization of partiality, nondeterminism, and reals
3. Some examples
4. Implementation and evaluation results
5. Conclusion

1. The classical logic without non-constructive reasoning principles:

Constructive Logic

1. The classical logic without non-constructive reasoning principles:
   - The law of excluded middle $P \vee \neg P$
   - Double negation elimination $\neg\neg P \to P$
   - $\cdots$

Constructive Logic

1. The classical logic without non-constructive reasoning principles:
   - The law of excluded middle $P \vee \neg P$
   - Double negation elimination $\neg\neg P \to P$
   - $\cdots$
2. Often introduce principles that contradict the classical logic:
   continuity principle "all $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ is continuous"

Constructive Logic

1. The classical logic without non-constructive reasoning principles:
   - The law of excluded middle $P \vee \neg P$
   - Double negation elimination $\neg\neg P \to P$
   - $\cdots$
2. Often introduce principles that contradict the classical logic:
   continuity principle "all $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ is continuous"
3. In consequence, proofs have to be done constructively:

Constructive Logic

1. The classical logic without non-constructive reasoning principles:
   - The law of excluded middle $P \vee \neg P$
   - Double negation elimination $\neg\neg P \to P$
   - $\cdots$
2. Often introduce principles that contradict the classical logic:
   continuity principle "all $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ is continuous"
3. In consequence, proofs have to be done constructively:
   - $P \vee Q$: have to specify if it is P or Q
   - $\exists x.\, P(x)$: have to construct x that satisfies $P(x)$

Constructive Logic

1. The classical logic without non-constructive reasoning principles:
   - The law of excluded middle $P \vee \neg P$
   - Double negation elimination $\neg\neg P \to P$
   - $\cdots$
2. Often introduce principles that contradict the classical logic:
   continuity principle "all $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ is continuous"
3. In consequence, proofs have to be done constructively:
   - $P \vee Q$: have to specify if it is P or Q
   - $\exists x.\, P(x)$: have to construct x that satisfies $P(x)$

The belief in the universal validity of the principle of the excluded third in mathematics is considered by the intuitionists as a phenomenon of the history of civilization of the same kind as the former belief in the rationality of π, or in the rotation of the firmament about the earth - L. E. J. Brouwer

Program Extraction

for any prime numbers $p_1 < \cdots < p_d$ *there exists a prime number* $p > p_d$

Program Extraction

for any prime numbers $p_1 < \cdots < p_d$ *there exists a prime number* $p > p_d$

1. Suppose any prime $p_1 < \cdots < p_d$

Program Extraction

for any prime numbers $p_1 < \cdots < p_d$ *there exists a prime number* $p > p_d$

1. Suppose any prime $p_1 < \cdots < p_d$
2. **Construct** a natural number p

Program Extraction

for any prime numbers $p_1 < \cdots < p_d$ *there exists a prime number* $p > p_d$

1. Suppose any prime $p_1 < \cdots < p_d$
2. **Construct** a natural number p
3. Prove (i) p is a prime number and (ii) $p > p_d$ holds

Program Extraction

for any prime numbers $p_1 < \cdots < p_d$ *there exists a prime number* $p > p_d$

1. Suppose any prime $p_1 < \cdots < p_d$
2. **Construct** a natural number p
3. Prove (i) p is a prime number and (ii) $p > p_d$ holds

The proof contains information on how to **construct** p given $p_1, \cdots, p_d$

Program Extraction

for any prime numbers $p_1 < \cdots < p_d$ *there exists a prime number* $p > p_d$

1. Suppose any prime $p_1 < \cdots < p_d$
2. **Construct** a natural number p
3. Prove (i) p is a prime number and (ii) $p > p_d$ holds

The proof contains information on how to **construct** p given $p_1, \cdots, p_d$

$=$ (extracting computational content) $\Rightarrow$

A *provably correct* computer program that **computes** p from $p_1, \cdots, p_d$

Program Extraction

for any prime numbers $p_1 < \cdots < p_d$ there exists a prime number $p > p_d$

1. Suppose any prime $p_1 < \cdots < p_d$
2. **Construct** a natural number p
3. Prove (i) p is a prime number and (ii) $p > p_d$ holds

The proof contains information on how to **construct** p given $p_1, \cdots, p_d$

$=$ (extracting computational content) $\Rightarrow$

A *provably correct* computer program that **computes** p from $p_1, \cdots, p_d$

- We need a framework where "proof"s become objects

Dependent Type Theory

- A foundation of mathematics...
- The foundation of mathematics???

Dependent Type Theory

- A foundation of mathematics...
- The foundation of mathematics???
- A language where we can do all: Construct and Prove (and compute)

Construction	Sets $A, B, C, \cdots$	Elements $a, b, c, \cdots$
Logic	Propositions $a \in A$	Proofs

Dependent Type Theory

- A foundation of mathematics...
- The foundation of mathematics???
- A language where we can do all: Construct and Prove (and compute)

Construction	Sets $A, B, C, \cdots$	Elements $a, b, c, \cdots$
Logic	Propositions $a \in A$	Proofs
Type Theory	Types	Terms

Dependent Type Theory

- A foundation of mathematics...
- The foundation of mathematics???
- A language where we can do all: Construct and Prove (and compute)

Construction	Sets $A, B, C, \cdots$	Elements $a, b, c, \cdots$
Logic	Propositions $a \in A$	Proofs
Type Theory	Types	Terms

- Proofs $\simeq$ λ-terms having transparent computational content

Problem Statement

- How do we introduce axiomatic real numbers in a dependent type theory?

Problem Statement

- How do we introduce axiomatic real numbers in a dependent type theory?
- Constructive type theory with type of classical propositions Prop closed under $\wedge, \tilde{\vee}, \forall, \tilde{\exists}, \to$:
  - $\forall(X : \mathsf{Type}).\ X \vee \neg X : \mathsf{Type}$ is invalid
  - $\forall(X : \mathsf{Type}).\ X\ \tilde{\vee}\ \neg X : \mathsf{Prop}$ valid
  - $X : \mathsf{Prop}$ is okay not to hold computational content

Problem Statement

- How do we introduce axiomatic real numbers in a dependent type theory?
- Constructive type theory with type of classical propositions Prop closed under $\wedge, \tilde{\vee}, \forall, \tilde{\exists}, \to$:
  - $\forall(X : \mathsf{Type}).\ X \vee \neg X : \mathsf{Type}$ is invalid
  - $\forall(X : \mathsf{Type}).\ X\ \tilde{\vee}\ \neg X : \mathsf{Prop}$ valid
  - $X : \mathsf{Prop}$ is okay not to hold computational content
- Axiomatic type
  - $X : \mathsf{Type}$

Problem Statement

- How do we introduce axiomatic real numbers in a dependent type theory?
- Constructive type theory with type of classical propositions Prop closed under $\wedge, \tilde{\vee}, \forall, \tilde{\exists}, \to$:
  - $\forall(X : \mathsf{Type}).\ X \vee \neg X : \mathsf{Type}$ is invalid
  - $\forall(X : \mathsf{Type}).\ X\ \tilde{\vee}\ \neg X : \mathsf{Prop}$ valid
  - $X : \mathsf{Prop}$ is okay not to hold computational content
- Axiomatic type
  - $X : \mathsf{Type}$
- algebra
  - $+ : X \to X \to X$

Problem Statement

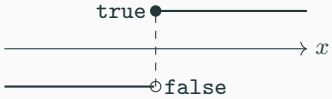
- How do we introduce axiomatic real numbers in a dependent type theory?
- Constructive type theory with type of classical propositions Prop closed under $\wedge, \tilde{\vee}, \forall, \tilde{\exists}, \to$:
  - $\forall(X : \mathsf{Type}).\ X \vee \neg X : \mathsf{Type}$ is invalid
  - $\forall(X : \mathsf{Type}).\ X\ \tilde{\vee}\ \neg X : \mathsf{Prop}$ valid
  - $X : \mathsf{Prop}$ is okay not to hold computational content
- Axiomatic type
  - $X : \mathsf{Type}$
- algebra
  - $+ : X \to X \to X$
- properties
  - $\mathsf{comm}_X : x + y = y + x$

Problem Statement

- How do we introduce axiomatic real numbers in a dependent type theory?
- Constructive type theory with type of classical propositions Prop closed under $\wedge, \tilde{\vee}, \forall, \tilde{\exists}, \to$:
  - $\forall(X : \mathsf{Type}).\ X \vee \neg X : \mathsf{Type}$ is invalid
  - $\forall(X : \mathsf{Type}).\ X\ \tilde{\vee}\ \neg X : \mathsf{Prop}$ valid
  - $X : \mathsf{Prop}$ is okay not to hold computational content
- Axiomatic type
  - $X : \mathsf{Type}$
- algebra
  - $+ : X \to X \to X$
- properties
  - $\mathsf{comm}_X : x + y = y + x$
- Classical axiomatization of reals R is invalid:

$$\forall(x : \mathsf{R}).\ x \geq 0 \vee x < 0$$

The sign function (any discontinuous function) is uncomputable

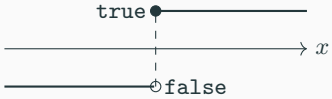


Problem Statement

- How do we introduce axiomatic real numbers in a dependent type theory?
- Constructive type theory with type of classical propositions Prop closed under $\wedge, \tilde{\vee}, \forall, \tilde{\exists}, \to$:
  - $\forall(X : \mathsf{Type}).\ X \vee \neg X : \mathsf{Type}$ is invalid
  - $\forall(X : \mathsf{Type}).\ X\ \tilde{\vee}\ \neg X : \mathsf{Prop}$ valid
  - $X : \mathsf{Prop}$ is okay not to hold computational content
- Axiomatic type
  - $X : \mathsf{Type}$
- algebra
  - $+ : X \to X \to X$
- properties
  - $\mathsf{comm}_X : x + y = y + x$
- Classical axiomatization of reals R is invalid:

  $$\forall(x : \mathsf{R}).\ x \geq 0 \vee x < 0$$

  The sign function (any discontinuous function) is uncomputable



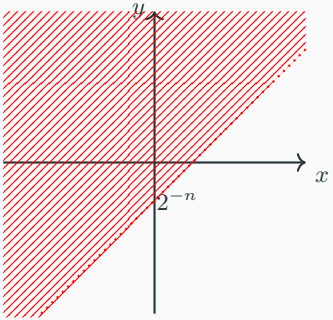
- Classical reals valid:

  $$\forall(x : \mathsf{R}).\ x \geq 0\ \tilde{}\ \vee\ x < 0$$

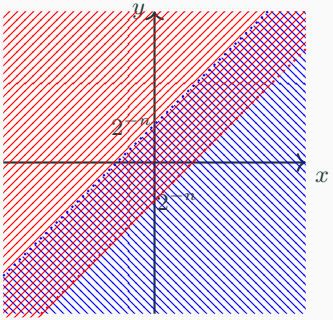
  but cannot do any effective reasoning

- $x < y$ diverges when $x = y$.

- $x < y$ diverges when $x = y$.
- $x < y + 2^{-n}$ diverges when $x = y + 2^{-n}$

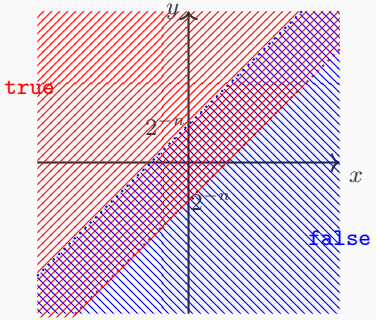


- $x < y$ diverges when $x = y$.
- $x < y + 2^{-n}$ diverges when $x = y + 2^{-n}$
- $y < x + 2^{-n}$ diverges when $y = x + 2^{-n}$



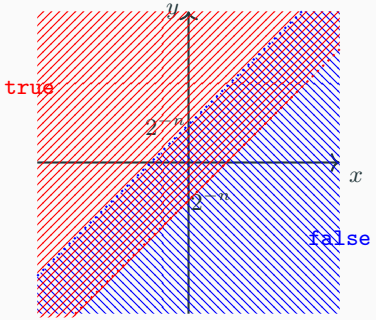
Nondeterministic COMPUTATION

- $x < y$ diverges when $x = y$.
- $x < y + 2^{-n}$ diverges when $x = y + 2^{-n}$
- $y < x + 2^{-n}$ diverges when $y = x + 2^{-n}$
- Test $x < y + 2^{-n}$ and $y < x + 2^{-n}$ in parallel
  Return `true` when $x < y + 2^{-n}$
  Return `false` when $y < x + 2^{-n}$



Nondeterministic COMPUTATION

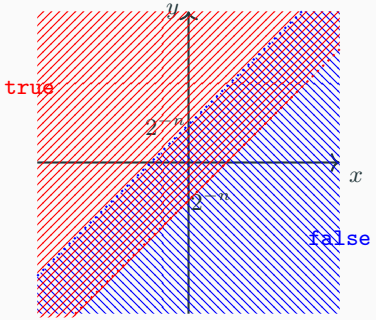
- $x < y$ diverges when $x = y$.
- $x < y + 2^{-n}$ diverges when $x = y + 2^{-n}$
- $y < x + 2^{-n}$ diverges when $y = x + 2^{-n}$
- Test $x < y + 2^{-n}$ and $y < x + 2^{-n}$ in parallel
  Return `true` when $x < y + 2^{-n}$
  Return `false` when $y < x + 2^{-n}$



$$x <_n y = \begin{cases} \texttt{true} & x \le y - 2^{-n}, \\ \texttt{true or false} & y - 2^{-n} < x < y + 2^{-n}, \\ \texttt{false} & y \le x - 2^{-n}. \end{cases}$$

Nondeterministic COMPUTATION

- $x < y$ diverges when $x = y$.
- $x < y + 2^{-n}$ diverges when $x = y + 2^{-n}$
- $y < x + 2^{-n}$ diverges when $y = x + 2^{-n}$
- Test $x < y + 2^{-n}$ and $y < x + 2^{-n}$ in parallel
  Return `true` when $x < y + 2^{-n}$
  Return `false` when $y < x + 2^{-n}$

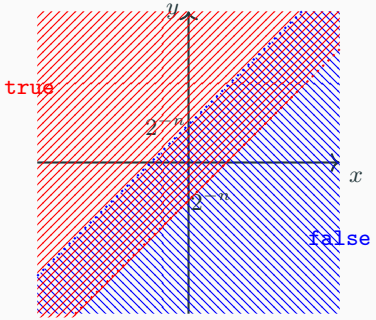


$$x <_n y = \begin{cases} \texttt{true} & x \leq y - 2^{-n}, \\ \texttt{true or false} & y - 2^{-n} < x < y + 2^{-n}, \\ \texttt{false} & y \leq x - 2^{-n}. \end{cases}$$

- Partiality and Nondeterminism need to be expressible in logical counterpart

Nondeterministic COMPUTATION

- $x < y$ diverges when $x = y$.
- $x < y + 2^{-n}$ diverges when $x = y + 2^{-n}$
- $y < x + 2^{-n}$ diverges when $y = x + 2^{-n}$
- Test $x < y + 2^{-n}$ and $y < x + 2^{-n}$ in parallel
  Return `true` when $x < y + 2^{-n}$
  Return `false` when $y < x + 2^{-n}$



$$x <_n y = \begin{cases} \texttt{true} & x \leq y - 2^{-n}, \\ \texttt{true or false} & y - 2^{-n} < x < y + 2^{-n}, \\ \texttt{false} & y \leq x - 2^{-n}. \end{cases}$$

- Partiality and Nondeterminism need to be expressible in logical counterpart
- Expressive enough for

$$\forall (x, y : \mathsf{R}).\ x < y \quad \textit{“is semi-decidable”}$$

- Expressive enough for

$$\forall (x, y : \mathsf{R}).\ x < y + 2^{-n} \quad \textit{“nondeterministically OR”}\ y < x + 2^{-n}$$

1. Intro: constructive type theory + why axiomatization of reals not trivial
2. **Axiomatization of partiality, nondeterminism, and reals**
3. Some examples
4. Implementation and evaluation results
5. Conclusion

Nondeterministic Types

Motivation:

- Given two semi-decidable propositions P, Q
- Given a promise that at least one of the two holds (without knowing exactly which one does)
- We can choose one nondeterministically

Nondeterministic Types

Motivation:

- Given two semi-decidable propositions P, Q
- Given a promise that at least one of the two holds (without knowing exactly which one does)
- We can choose one nondeterministically

Axiom (Kleene type)

- K is a type with three known elements $\texttt{true}, \texttt{false}, \bot : \mathsf{K}$

Nondeterministic Types

Motivation:

- Given two semi-decidable propositions P, Q
- Given a promise that at least one of the two holds (without knowing exactly which one does)
- We can choose one nondeterministically

Axiom (Kleene type)

- K is a type with three known elements $\texttt{true}, \texttt{false}, \bot : \mathsf{K}$
- Define $s\downarrow$ to denote $(s = \texttt{true}) : \mathsf{Prop}$

Definition

$\mathsf{semi}(P : \mathsf{Prop}) :\equiv \exists(s : \mathsf{K}).\ P = s\downarrow$

Axiom (Kleene type)

- K is a type with two known elements $\texttt{true}, \texttt{false} : \mathsf{K}$
- Define $s\downarrow$ to denote $(s = \texttt{true}) : \mathsf{Prop}$

Definition

$\mathsf{semi}(P : \mathsf{Prop}) :\equiv \exists(s : \mathsf{K}).\ P = s\downarrow$

Example

- $\forall(n, m : \mathsf{N}).\ \mathsf{semi}(n = m)$
- Define $\mathsf{dec}(P) :\equiv P \vee \neg P$. Then, $\mathsf{dec}(P) \to \mathsf{semi}(P)$
- $\mathsf{semi}(P) \to \mathsf{semi}(\neg P) \to \mathsf{dec}(P)$?

Axiom (Nondeterminism Monad)

- There is a moand $(\mathsf{M} : \mathsf{Type} \to \mathsf{Type}, \eta^{\mathsf{M}}, \mu^{\mathsf{M}}, \mathsf{lift}^{\mathsf{M}})$
- $\mathsf{M}X$ denotes results of nondeterministic computation in X

Axiom (Nondeterminism Monad)

- There is a moand $(\mathsf{M} : \mathsf{Type} \to \mathsf{Type}, \eta^{\mathsf{M}}, \mu^{\mathsf{M}}, \mathsf{lift}^{\mathsf{M}})$
- $\mathsf{M}X$ denotes results of nondeterministic computation in X
- such that

$$\mathsf{select} : \forall(x, y : \mathsf{K}).\ (x \downarrow\ \tilde{\vee}\ y \downarrow) \to \mathsf{M}\big(x \downarrow \vee y \downarrow\big)$$

Axiom (Nondeterminism Monad)

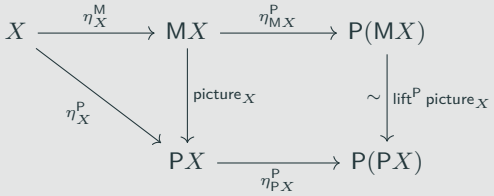
- There is a moand $(\mathsf{M} : \mathsf{Type} \to \mathsf{Type}, \eta^{\mathsf{M}}, \mu^{\mathsf{M}}, \mathsf{lift}^{\mathsf{M}})$
- $\mathsf{M}X$ denotes results of nondeterministic computation in X
- such that

$$\mathsf{select} : \forall(x, y : \mathsf{K}).\ (x\downarrow\ \tilde{\vee}\ y\downarrow) \to \mathsf{M}\big(x\downarrow \vee y\downarrow\big)$$

- and is submonoidal of the **classsical nonempty powerset** monad

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

where the submoindal natural transformation $\mathsf{picture}$ satisfies



Axiom (Nondeterminism Monad)

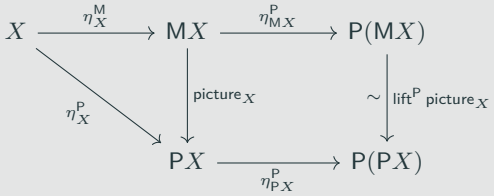
- There is a moand $(\mathsf{M} : \mathsf{Type} \to \mathsf{Type}, \eta^{\mathsf{M}}, \mu^{\mathsf{M}}, \mathsf{lift}^{\mathsf{M}})$
- $\mathsf{M}X$ denotes results of nondeterministic computation in X
- such that

$$\mathsf{select} : \forall(x, y : \mathsf{K}).\ (x \downarrow\ \tilde{\vee}\ y \downarrow) \to \mathsf{M}\big(x \downarrow \vee y \downarrow\big)$$

- and is submonoidal of the **classsical nonempty powerset** monad

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

where the submoindal natural transformation $\mathsf{picture}$ satisfies



- and admits subsingleton elimination

$$\big(\forall(x, y : X).\ x = y\big) \to (X \simeq \mathsf{M}X)$$

Definition (Nondeterminism Logic)

- Let $\mathring{\exists}(a : A).\ B(a)$ be the nondeterministic type $\mathsf{M}\big(\exists(a : A).\ B(a)\big)$
- Let $A \mathbin{\mathring{\vee}} B$ be the nondeterministic type $\mathsf{M}\big(A \vee B\big)$

Example (nondeterministic branching)

- Suppose $f : A \to \exists(x : C).\ P(x)$
- Suppose $g : B \to \exists(x : C).\ P(x)$

Definition (Nondeterminism Logic)

- Let $\mathring{\exists}(a : A).\ B(a)$ be the nondeterministic type $\mathsf{M}\big(\exists(a : A).\ B(a)\big)$
- Let $A \mathbin{\mathring{\vee}} B$ be the nondeterministic type $\mathsf{M}\big(A \vee B\big)$

Example (nondeterministic branching)

- Suppose $f : A \to \exists(x : C).\ P(x)$
- Suppose $g : B \to \exists(x : C).\ P(x)$
- Case distinction yields $k : A \vee B \to \exists(x : C).\ P(x)$

Definition (Nondeterminism Logic)

- Let $\mathring{\exists}(a : A).\ B(a)$ be the nondeterministic type $\mathsf{M}\big(\exists(a : A).\ B(a)\big)$
- Let $A \mathbin{\mathring{\vee}} B$ be the nondeterministic type $\mathsf{M}\big(A \vee B\big)$

Example (nondeterministic branching)

- Suppose $f : A \to \exists(x : C).\ P(x)$
- Suppose $g : B \to \exists(x : C).\ P(x)$
- Case distinction yields $k : A \vee B \to \exists(x : C).\ P(x)$
- Injection and lifting yields $\mathsf{lift}^{\mathsf{M}} k : A \mathbin{\mathring{\vee}} B \to \mathring{\exists}(x : C).\ P(x)$

Definition (Nondeterminism Logic)

- Let $\mathring{\exists}(a : A).\ B(a)$ be the nondeterministic type $\mathsf{M}\big(\exists(a : A).\ B(a)\big)$
- Let $A \mathbin{\mathring{\vee}} B$ be the nondeterministic type $\mathsf{M}\big(A \vee B\big)$

Example (nondeterministic branching)

- Suppose $f : A \to \exists(x : C).\ P(x)$
- Suppose $g : B \to \exists(x : C).\ P(x)$
- Case distinction yields $k : A \vee B \to \exists(x : C).\ P(x)$
- Injection and lifting yields $\mathsf{lift}^{\mathsf{M}} k : A \mathbin{\mathring{\vee}} B \to \mathring{\exists}(x : C).\ P(x)$
- If $A \mathbin{\mathring{\vee}} B$ is known, we have $\mathring{\exists}(x : C).\ P(x)$

Nondeterministic Choose

Motivation:

- Given two semi-decidable propositions P, Q
- Given a promise that at least one of the two holds (without knowing exactly which one does)
- We can choose one nondeterministically

Nondeterministic Choose

Motivation:

- Given two semi-decidable propositions P, Q
- Given a promise that at least one of the two holds (without knowing exactly which one does)
- We can choose one nondeterministically

Axiom (nondeterministic select)

$\mathsf{select} : \forall(s_1, s_2 : \mathsf{K}).\ s_1 \downarrow\ \tilde{\vee}\ s_2 \downarrow \to\ s_1 \downarrow\ \mathring{\vee}\ s_2 \downarrow$

Nondeterministic Choose

Motivation:

- Given two semi-decidable propositions P, Q
- Given a promise that at least one of the two holds (without knowing exactly which one does)
- We can choose one nondeterministically

Axiom (nondeterministic select)

select : $\forall(s_1, s_2 : \mathsf{K}).\ s_1 \downarrow\ \tilde{\vee}\ s_2 \downarrow \rightarrow\ \ s_1 \downarrow\ \mathring{\vee}\ s_2 \downarrow$

Definition

semi $P := \exists(s : \mathsf{K}).\ P \leftrightarrow s \downarrow$

Lemma

choose : $\forall(P, Q : \mathsf{Prop}).\ \mathit{semi}(P) \rightarrow \mathit{semi}(Q) \rightarrow P\ \tilde{\vee}\ Q \rightarrow\ \ P\ \mathring{\vee}\ Q$

Example (Dijkstra style guarded expressions)

$$\mathsf{case}\ p\ q\ r\ s:$$
$$\mathsf{semi}(p) \to \mathsf{semi}(q) \to p \mathbin{\tilde{\vee}} q \to (p \to r) \to (q \to s) \to (p \wedge r) \mathbin{\mathring{\vee}} (q \wedge s)$$

Proof

Qed

Example (Dijkstra style guarded expressions)

$$\mathsf{case}\ p\ q\ r\ s:$$
$$\mathsf{semi}(p) \to \mathsf{semi}(q) \to p \mathbin{\tilde{\vee}} q \to (p \to r) \to (q \to s) \to (p \wedge r) \mathbin{\mathring{\vee}} (q \wedge s)$$

Proof

- choose yields $p \mathbin{\mathring{\vee}} q$

Qed

Example (Dijkstra style guarded expressions)

$$\mathsf{case}\ p\ q\ r\ s :$$
$$\mathsf{semi}(p) \to \mathsf{semi}(q) \to p \mathbin{\tilde{\vee}} q \to (p \to r) \to (q \to s) \to (p \wedge r) \mathbin{\mathring{\vee}} (q \wedge s)$$

Proof

- choose yields $p \mathbin{\mathring{\vee}} q$
- We can construct a term $t : p \vee q \to (p \wedge r) \vee (q \wedge s)$

Qed

Example (Dijkstra style guarded expressions)

$$\mathsf{case}\ p\ q\ r\ s :$$
$$\mathsf{semi}(p) \to \mathsf{semi}(q) \to p \mathbin{\tilde{\vee}} q \to (p \to r) \to (q \to s) \to (p \wedge r) \mathbin{\mathring{\vee}} (q \wedge s)$$

Proof

- choose yields $p \mathbin{\mathring{\vee}} q$
- We can construct a term $t : p \vee q \to (p \wedge r) \vee (q \wedge s)$
- We can lift it to get $p \mathbin{\mathring{\vee}} q \to (p \wedge r) \mathbin{\mathring{\vee}} (q \wedge s)$

Qed

Nondeterministic Types: Examples

Example

$\forall(P : \mathsf{Prop}).\ \mathsf{semi}(P) \to \mathsf{semi}(\neg P) \to \mathsf{dec}(P)$

Proof

Qed

Nondeterministic Types: Examples

Example

$\forall(P : \mathsf{Prop}).\ \mathsf{semi}(P) \to \mathsf{semi}(\neg P) \to \mathsf{dec}(P)$

Proof

- $\mathsf{LEM}(P) : P\ \tilde{\vee}\ \neg P$

Qed

Nondeterministic Types: Examples

Example

$\forall(P : \mathsf{Prop}).\ \mathsf{semi}(P) \to \mathsf{semi}(\neg P) \to \mathsf{dec}(P)$

Proof

- $\mathsf{LEM}(P) : P \mathbin{\tilde{\vee}} \neg P$
- Let $p_1 : \mathsf{semi}(P)$ and $p_2 : \mathsf{semi}(\neg P)$

Qed

Nondeterministic Types: Examples

Example

$\forall(P : \mathsf{Prop}).\ \mathsf{semi}(P) \to \mathsf{semi}(\neg P) \to \mathsf{dec}(P)$

Proof

- $\mathsf{LEM}(P) : P \mathbin{\tilde{\vee}} \neg P$
- Let $p_1 : \mathsf{semi}(P)$ and $p_2 : \mathsf{semi}(\neg P)$
- choose $P\ \neg P\ p_1\ p_2\ (\mathsf{LEM}(P)) : P \mathbin{\mathring{\vee}} \neg P$

Qed

Example

$\forall(P : \mathsf{Prop}).\ \mathsf{semi}(P) \to \mathsf{semi}(\neg P) \to \mathsf{dec}(P)$

Proof

- $\mathsf{LEM}(P) : P \mathbin{\tilde{\vee}} \neg P$
- Let $p_1 : \mathsf{semi}(P)$ and $p_2 : \mathsf{semi}(\neg P)$
- choose $P\ \neg P\ p_1\ p_2\ (\mathsf{LEM}(P)) : P \mathbin{\mathring{\vee}} \neg P$
- $P \vee \neg P$ is subsingleton

Qed

Nondeterministic Types: Examples

Example

$\forall(P : \mathsf{Prop}).\ \mathsf{semi}(P) \to \mathsf{semi}(\neg P) \to \mathsf{dec}(P)$

Proof

- $\mathsf{LEM}(P) : P \mathbin{\tilde{\vee}} \neg P$
- Let $p_1 : \mathsf{semi}(P)$ and $p_2 : \mathsf{semi}(\neg P)$
- choose $P\ \neg P\ p_1\ p_2\ (\mathsf{LEM}(P)) : P \mathbin{\mathring{\vee}} \neg P$
- $P \vee \neg P$ is subsingleton
- extraction axiom yields $P \vee \neg P$

Qed

Axiom (Reals $\mathbb{R}$)

- R is a type with two known elements $0, 1 : \mathsf{R}$.
- $+, -, \times, ^{-1}, >$ are term constants
- The axioms are inhabited

	field axioms	A_1	$0 \neq 1$
A_2	$x + y = y + x$	A_3	$x + y + z = x + (y + z)$
A_4	$x + 0 = x$	A_5	$x - x = 0$
A_6	$x \times y = y \times x$	A_7	$x \times y \times z = x \times (y \times z)$
A_8	$x \times 1 = x$	A_9	$\forall (t : x \neq 0).\ x \times t^{-1} = 1$
A_{10}	$x \times (y + z) = x \times y + x \times z$		order axioms
A_{11}	$x = y \ \tilde{\vee}\ x < y \ \tilde{\vee}\ y < x$	A_{12}	$x < y \to y < z \to x < z$
A_{13}	$x < y \to \neg y < x$	A_{14}	$x < y \to x + z < y + z$
A_{15}	$0 < z \to x < y \to z \times x < z \times y$	A_{16}	$\mathsf{semi}(x < y)$
	metric completeness	A_{17}	"something about limit"

1. Intro: constructive type theory + why axiomatization of reals not trivial
2. Axiomatization of partiality, nondeterminism, and reals
3. **Some examples**
4. Implementation and evaluation results
5. Conclusion

Example

SoftComp : $\forall(x, y, \varepsilon : \mathsf{R}).\ 0 < \varepsilon \to (y - \varepsilon < x \mathbin{\mathring{\vee}} x - \varepsilon < y)$

Proof

Qed

Example

SoftComp : $\forall(x, y, \varepsilon : \mathsf{R}).\ 0 < \varepsilon \to (y - \varepsilon < x \mathbin{\mathring{\vee}} x - \varepsilon < y)$

Proof

- $t : y - \varepsilon < x \mathbin{\tilde{\vee}} x - \varepsilon < y$ form weak total order

Qed

Example

SoftComp : $\forall(x, y, \varepsilon : \mathsf{R}).\ 0 < \varepsilon \to (y - \varepsilon < x \mathbin{\mathring{\vee}} x - \varepsilon < y)$

Proof

- $t : y - \varepsilon < x \mathbin{\tilde{\vee}} x - \varepsilon < y$ form weak total order
- $a : \mathsf{semi}(y - \varepsilon < x)$ and $b : \mathsf{semi}(y - \varepsilon < x)$ are from axioms

Qed

Example

SoftComp : $\forall(x, y, \varepsilon : \mathsf{R}).\ 0 < \varepsilon \to (y - \varepsilon < x \mathbin{\mathring{\vee}} x - \varepsilon < y)$

Proof

- $t : y - \varepsilon < x \mathbin{\tilde{\vee}} x - \varepsilon < y$ form weak total order
- $a : \mathsf{semi}(y - \varepsilon < x)$ and $b : \mathsf{semi}(y - \varepsilon < x)$ are from axioms
- choose $a\ b\ t$ is a proof for $(y - \varepsilon < x \mathbin{\mathring{\vee}} x - \varepsilon < y)$

Qed

Safe order comparison

Example

$\forall(x : \mathsf{R})\ x \neq 0 \to 0 < x \vee x < 0$

Proof

Qed

Example

$\forall(x : \mathsf{R})\ x \neq 0 \to 0 < x \vee x < 0$

Proof

- From weak total order, $0 < 0\ \tilde{\vee}\ 0 < x$

Qed

Example

$\forall(x : \mathsf{R})\ x \neq 0 \to 0 < x \vee x < 0$

Proof

- From weak total order, $0 < 0 \,\tilde{\vee}\, 0 < x$
- From choose and semi, we have $0 < x \,\mathring{\vee}\, x < 0$

Qed

Safe order comparison

Example

$\forall(x : \mathsf{R})\ x \neq 0 \to 0 < x \vee x < 0$

Proof

- From weak total order, $0 < 0 \mathbin{\tilde{\vee}} 0 < x$
- From choose and semi, we have $0 < x \mathbin{\mathring{\vee}} x < 0$
- $0 < x \vee x < 0$ sub-singleton

Qed

Safe order comparison

Example

$\forall(x : \mathsf{R})\ x \neq 0 \to 0 < x \vee x < 0$

Proof

- From weak total order, $0 < 0 \,\tilde{\vee}\, 0 < x$
- From choose and semi, we have $0 < x \,\mathring{\vee}\, x < 0$
- $0 < x \vee x < 0$ sub-singleton
- Extraction axiom yields $0 < x \vee x < 0$

Qed

Advanced Examples

- Metric completeness axiomatized as "every effective Cauchy sequence constructive has limit"
- Mixing it with the nondeterminism, can prove
  - construction of maximum

$$\forall(x, y : \mathsf{R}).\ \exists(z : \mathsf{R}).\ x < y \to z = y \wedge (\neg x < y) \to z = x$$

  - construction of square root

$$\forall(x : \mathsf{R}).\ 0 \leq x \to \exists(y : \mathsf{R}).\ y \times y = x$$

  - constructive Intermediate Value Theorem

$$\forall(f : \mathsf{R} \to \mathsf{R}).\ f(0) < 0 < f(1) \to (\tilde{\exists}!(x : \mathsf{R}).\ f(x) = 0) \to \exists!(x : \mathsf{R}).\ f(x) = 0)$$

    assuming the continuity principle

1. Intro: constructive type theory + why axiomatization of reals not trivial
2. Axiomatization of partiality, nondeterminism, and reals
3. Some examples
4. **Implementation and evaluation results**
5. Conclusion

Program Extraction

When we prove a statement

$$\forall(x : \mathsf{R}).\ \exists(y : \mathsf{R}).\ P\ x\ y$$

we get a computable function $f : \mathbb{R} \to \mathbb{R}$ computing numbers with the property P.

Program Extraction

When we prove a statement

$$\forall(x : \mathsf{R}).\ \exists(y : \mathsf{R}).\ P\ x\ y$$

we get a computable function $f : \mathbb{R} \to \mathbb{R}$ computing numbers with the property P.

When we prove

$$\forall(x : \mathsf{R}).\ \mathsf{M}\exists(y : \mathsf{R}).\ P\ x\ y$$

we get a multivalued function $f : \mathbb{R} \rightrightarrows \mathbb{R}$.

Program Extraction

When we prove a statement

$$\forall(x : \mathsf{R}).\ \exists(y : \mathsf{R}).\ P\ x\ y$$

we get a computable function $f : \mathbb{R} \to \mathbb{R}$ computing numbers with the property P.

When we prove

$$\forall(x : \mathsf{R}).\ \mathsf{M}\exists(y : \mathsf{R}).\ P\ x\ y$$

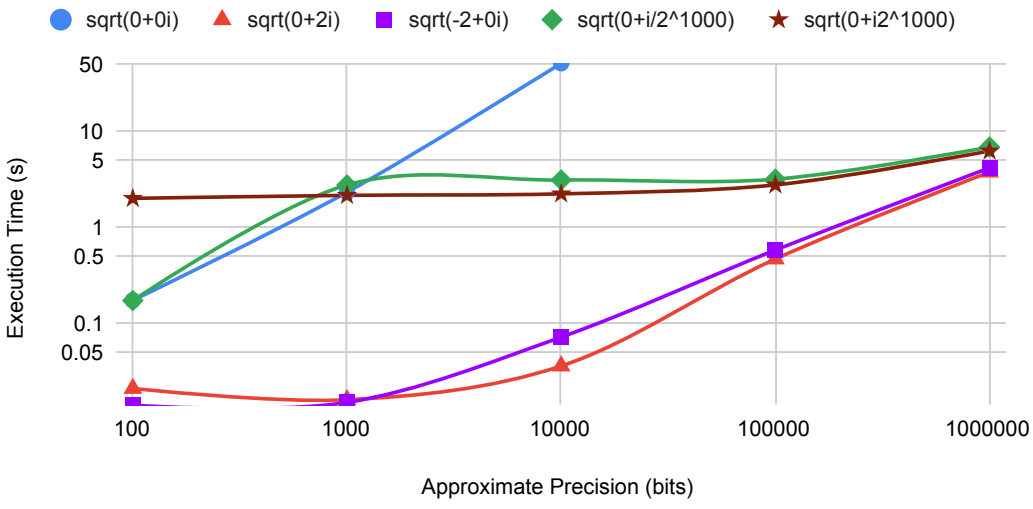
we get a multivalued function $f : \mathbb{R} \rightrightarrows \mathbb{R}$.
In the Coq implemtation this is realized by mapping R to AERN's datatype `CReal` and axiomatized operations to primitive operations in AERN.

Quality of programs: Execution speed

Benchmark		Average execution time (s)			
Formula	Accuracy	Extracted	Hand-written	Native	iRRAM
$\max(0, \pi - \pi)$	10^6 bits	3.5	3.8	3.8	1.59
$\sqrt{2}$	10^6 bits	0.72	0.70	0.40	0.62
$\sqrt{\sqrt{2}}$	10^6 bits	1.52	1.38	0.85	1.15
$x - 0.5 = 0$	10^3 bits	1.44	0.32	—	0.03
$x(2 - x) - 0.5 = 0$	10^3 bits	2.02	0.35	—	0.04
$\sqrt{x + 0.5} - 1 = 0$	10^3 bits	12.9	2.35	—	0.29

(i7-4710MQ CPU, 16GB RAM, Ubuntu 18.04, Haskell Stackage LTS 17.2)

Quality of programs: Execution speed



(i7-4710MQ CPU, 16GB RAM, Ubuntu 18.04, Haskell Stackage LTS 17.2)

1. Intro: constructive type theory + why axiomatization of reals not trivial
2. Axiomatization of partiality, nondeterminism, and reals
3. Some examples
4. Implementation and evaluation results
5. **Conclusion**

Summary

- We specified a set of axioms for real numbers and nondeterministic computation over them in a constructive type theory
- proved various - simple but interesting to advanced - examples showing the usefulness of this axiomatization
- (omitted from the talk) proved the soundness by extending the realizability interpretation of the type theory
- (omitted from the talk) extended the nondeterminism monad with nondeterministic choice principle
- (omitted from the talk) proved the nondeterministic existence of complex square roots
- implemented them in Coq and extracted to AERN/Haskel programs

Available in detail in:


Michal Konečný, Sewon Park, and Holger Thies.
Axiomatic Reals and Certified Efficient Exact Real Computation.
WoLLIC 2021. Springer, Cham, 2021.


Michal Konečný, Sewon Park, and Holger Thies.
Extracting efficient exact real number computation from proofs in constructive type theory.
Arxiv preprint (arXiv:2202.00891).