Computational content of proofs involving coinduction

Helmut Schwichtenberg
(j.w.w. Kenji Miyamoto and Fredrik Nordvall Forsberg)

Mathematisches Institut, LMU, München

Advances in Proof Theory, Universität Bern,
13.-14. Dezember 2013

Computable functionals

Arguments of any finite type, not only numbers and functions.

- Principle of finite support. If $\mathcal{H}(\Phi)$ is defined with value n, then there is a finite approximation Φ_0 of Φ such that $\mathcal{H}(\Phi_0)$ is defined with value n.
- Monotonicity principle. If $\mathcal{H}(\Phi)$ is defined with value n and Φ' extends Φ, then also $\mathcal{H}(\Phi')$ is defined with value n.
- Effectivity principle. An object is computable iff its set of finite approximations is (primitive) recursively enumerable (or equivalently, Σ^0_1-definable).

Computable functionals

Arguments of any finite type, not only numbers and functions.

- Principle of finite support. If $\mathcal{H}(\Phi)$ is defined with value n, then there is a finite approximation Φ_0 of Φ such that $\mathcal{H}(\Phi_0)$ is defined with value n.
- Monotonicity principle. If $\mathcal{H}(\Phi)$ is defined with value n and Φ' extends Φ, then also $\mathcal{H}(\Phi')$ is defined with value n.
- Effectivity principle. An object is computable iff its set of finite approximations is (primitive) recursively enumerable (or equivalently, Σ^0_1-definable).

Computable functionals

Arguments of any finite type, not only numbers and functions.

- Principle of finite support. If $\mathcal{H}(\Phi)$ is defined with value n, then there is a finite approximation Φ_0 of Φ such that $\mathcal{H}(\Phi_0)$ is defined with value n.
- Monotonicity principle. If $\mathcal{H}(\Phi)$ is defined with value n and Φ' extends Φ, then also $\mathcal{H}(\Phi')$ is defined with value n.
- Effectivity principle. An object is computable iff its set of finite approximations is (primitive) recursively enumerable (or equivalently, Σ^0_1-definable).

Computable functionals

Arguments of any finite type, not only numbers and functions.

- Principle of finite support. If $\mathcal{H}(\Phi)$ is defined with value n, then there is a finite approximation Φ_0 of Φ such that $\mathcal{H}(\Phi_0)$ is defined with value n.
- Monotonicity principle. If $\mathcal{H}(\Phi)$ is defined with value n and Φ' extends Φ, then also $\mathcal{H}(\Phi')$ is defined with value n.
- Effectivity principle. An object is computable iff its set of finite approximations is (primitive) recursively enumerable (or equivalently, Σ^0_1-definable).

Computable functionals

Arguments of any finite type, not only numbers and functions.

- Principle of finite support. If $\mathcal{H}(\Phi)$ is defined with value n, then there is a finite approximation Φ_0 of Φ such that $\mathcal{H}(\Phi_0)$ is defined with value n.
- Monotonicity principle. If $\mathcal{H}(\Phi)$ is defined with value n and Φ' extends Φ, then also $\mathcal{H}(\Phi')$ is defined with value n.
- Effectivity principle. An object is computable iff its set of finite approximations is (primitive) recursively enumerable (or equivalently, Σ^0_1-definable).

Tokens, consistency and entailment at base types

Types

- Base types ι: free algebras, given by constructors (e.g. 0, S).
- Function types: $\rho \to \sigma$.

Example: $\iota := \mathbf{D}$ (derivations, or binary trees), by constructors $\circ$ (leaf, or nil) and $C \colon \mathbf{D} \to \mathbf{D} \to \mathbf{D}$ (branch, or cons).

- Token $a^{\mathbf{D}}$: $\circ$, $C{*}\circ$, $C\circ{*}$, $C(C{*}\circ)\circ$.
- $U^{\mathbf{D}} := \{a_1, \dots, a_n\}$ consistent if
  - all a_i start with the same constructor,
  - (proper) tokens at j-th argument positions are consistent (example: $\{C{*}\circ, C\circ{*}\}$).
- $U^{\mathbf{D}} \vdash a$ (entails) if
  - all $a_i \in U$ and a start with the same constructor,
  - (proper) tokens at j-th argument positions of a_i entail j-th argument of a (example: $\{C{*}\circ, C\circ{*}\} \vdash C\circ\circ$).

An ideal x^ρ is a (possibly infinite) set of tokens which is

- consistent and
- closed under entailment.

Tokens, consistency and entailment at base types

Types

- Base types ι: free algebras, given by constructors (e.g. 0, S).
- Function types: $\rho \to \sigma$.

Example: $\iota := \mathbf{D}$ (derivations, or binary trees), by constructors $\circ$ (leaf, or nil) and $C \colon \mathbf{D} \to \mathbf{D} \to \mathbf{D}$ (branch, or cons).

- Token $a^{\mathbf{D}}$: $\circ$, $C{*}\circ$, $C\circ{*}$, $C(C{*}\circ)\circ$.
- $U^{\mathbf{D}} := \{a_1, \dots, a_n\}$ consistent if
  - all a_i start with the same constructor,
  - (proper) tokens at j-th argument positions are consistent (example: $\{C{*}\circ, C\circ{*}\}$).
- $U^{\mathbf{D}} \vdash a$ (entails) if
  - all $a_i \in U$ and a start with the same constructor,
  - (proper) tokens at j-th argument positions of a_i entail j-th argument of a (example: $\{C{*}\circ, C\circ{*}\} \vdash C\circ\circ$).

An ideal x^{ρ} is a (possibly infinite) set of tokens which is

- consistent and
- closed under entailment.

Tokens, consistency and entailment at base types

Types

- Base types ι: free algebras, given by constructors (e.g. 0, S).
- Function types: $\rho \to \sigma$.

Example: $\iota := \mathbf{D}$ (derivations, or binary trees), by constructors $\circ$ (leaf, or nil) and $C\colon \mathbf{D} \to \mathbf{D} \to \mathbf{D}$ (branch, or cons).

- Token $a^{\mathbf{D}}$: $\circ$, $C{*}\circ$, $C{\circ}{*}$, $C(C{*}\circ)\circ$.
- $U^{\mathbf{D}} := \{a_1, \dots, a_n\}$ consistent if
  - all a_i start with the same constructor,
  - (proper) tokens at j-th argument positions are consistent (example: $\{C{*}\circ, C{\circ}{*}\}$).
- $U^{\mathbf{D}} \vdash a$ (entails) if
  - all $a_i \in U$ and a start with the same constructor,
  - (proper) tokens at j-th argument positions of a_i entail j-th argument of a (example: $\{C{*}\circ, C{\circ}{*}\} \vdash C{\circ}{\circ}$).

An ideal x^{ρ} is a (possibly infinite) set of tokens which is

- consistent and
- closed under entailment.

Tokens, consistency and entailment at base types

Types

- Base types ι: free algebras, given by constructors (e.g. 0, S).
- Function types: $\rho \to \sigma$.

Example: $\iota := \mathbf{D}$ (derivations, or binary trees), by constructors $\circ$ (leaf, or nil) and $C\colon \mathbf{D} \to \mathbf{D} \to \mathbf{D}$ (branch, or cons).

- Token $a^{\mathbf{D}}$: $\circ$, $C{*}\circ$, $C\circ{*}$, $C(C{*}\circ)\circ$.
- $U^{\mathbf{D}} := \{a_1, \dots, a_n\}$ consistent if
  - all a_i start with the same constructor,
  - (proper) tokens at j-th argument positions are consistent (example: $\{C{*}\circ, C\circ{*}\}$).
- $U^{\mathbf{D}} \vdash a$ (entails) if
  - all $a_i \in U$ and a start with the same constructor,
  - (proper) tokens at j-th argument positions of a_i entail j-th argument of a (example: $\{C{*}\circ, C\circ{*}\} \vdash C\circ\circ$).

An ideal x^{ρ} is a (possibly infinite) set of tokens which is

- consistent and
- closed under entailment.

Tokens, consistency and entailment at base types

Types

- Base types ι: free algebras, given by constructors (e.g. 0, S).
- Function types: $\rho \to \sigma$.

Example: $\iota := \mathbf{D}$ (derivations, or binary trees), by constructors $\circ$ (leaf, or nil) and $C \colon \mathbf{D} \to \mathbf{D} \to \mathbf{D}$ (branch, or cons).

- Token $a^{\mathbf{D}}$: $\circ$, $C{*}\circ$, $C\circ{*}$, $C(C{*}\circ)\circ$.
- $U^{\mathbf{D}} := \{a_1, \dots, a_n\}$ consistent if
  - all a_i start with the same constructor,
  - (proper) tokens at j-th argument positions are consistent (example: $\{C{*}\circ, C\circ{*}\}$).
- $U^{\mathbf{D}} \vdash a$ (entails) if
  - all $a_i \in U$ and a start with the same constructor,
  - (proper) tokens at j-th argument positions of a_i entail j-th argument of a (example: $\{C{*}\circ, C\circ{*}\} \vdash C\circ\circ$).

An ideal x^ρ is a (possibly infinite) set of tokens which is

- consistent and
- closed under entailment.

Tokens, consistency and entailment at base types

Types

- Base types ι: free algebras, given by constructors (e.g. 0, S).
- Function types: $\rho \to \sigma$.

Example: $\iota := \mathbf{D}$ (derivations, or binary trees), by constructors $\circ$ (leaf, or nil) and $C \colon \mathbf{D} \to \mathbf{D} \to \mathbf{D}$ (branch, or cons).

- Token $a^{\mathbf{D}}$: $\circ$, $C{*}\circ$, $C\circ{*}$, $C(C{*}\circ)\circ$.
- $U^{\mathbf{D}} := \{a_1, \dots, a_n\}$ consistent if
  - all a_i start with the same constructor,
  - (proper) tokens at j-th argument positions are consistent (example: $\{C{*}\circ, C\circ{*}\}$).
- $U^{\mathbf{D}} \vdash a$ (entails) if
  - all $a_i \in U$ and a start with the same constructor,
  - (proper) tokens at j-th argument positions of a_i entail j-th argument of a (example: $\{C{*}\circ, C\circ{*}\} \vdash C\circ\circ$).

An ideal x^{ρ} is a (possibly infinite) set of tokens which is

- consistent and
- closed under entailment.

Tokens, consistency and entailment at base types

Types

- Base types ι: free algebras, given by constructors (e.g. 0, S).
- Function types: $\rho \to \sigma$.

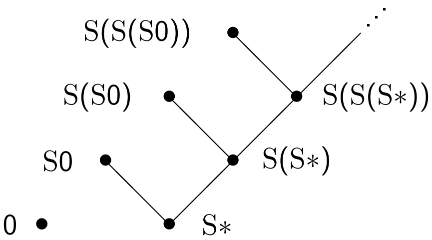
Example: $\iota := \mathbf{D}$ (derivations, or binary trees), by constructors $\circ$ (leaf, or nil) and $C\colon \mathbf{D} \to \mathbf{D} \to \mathbf{D}$ (branch, or cons).

- Token $a^{\mathbf{D}}$: $\circ$, $C{*}\circ$, $C\circ{*}$, $C(C{*}\circ)\circ$.
- $U^{\mathbf{D}} := \{a_1, \dots, a_n\}$ consistent if
  - all a_i start with the same constructor,
  - (proper) tokens at j-th argument positions are consistent (example: $\{C{*}\circ, C\circ{*}\}$).
- $U^{\mathbf{D}} \vdash a$ (entails) if
  - all $a_i \in U$ and a start with the same constructor,
  - (proper) tokens at j-th argument positions of a_i entail j-th argument of a (example: $\{C{*}\circ, C\circ{*}\} \vdash C\circ\circ$).

An ideal x^{ρ} is a (possibly infinite) set of tokens which is

- consistent and
- closed under entailment.

Tokens and entailment for **N**



$\{a\} \vdash b$ iff there is a path from a (up) to b (down).

Total and cototal ideals of base type

An ideal x^ι is cototal if every constructor tree $P(*) \in x$ has a "$\succ_1$-predecessor" $P(\mathrm{C}\vec{*}) \in x$; it is total if it is cototal and the relation $\succ_1$ on x is well-founded.

Examples. **N**:

- Every total ideal is the deductive closure of a token $\mathrm{S}(\mathrm{S}\ldots(\mathrm{S}0)\ldots)$. The set of all tokens $\mathrm{S}(\mathrm{S}\ldots(\mathrm{S}*)\ldots)$ is a cototal ideal.

D (derivations):

- Total ideal $\sim$ finite derivation.
- Cototal ideal $\sim$ finite or infinite "locally correct" derivation [Mints 78].
- Arbitrary ideal $\sim$ incomplete derivation, with "holes".

Total and cotoval ideals of base type

An ideal x^ι is cotoval if every constructor tree $P(*) \in x$ has a "$\succ_1$-predecessor" $P(\mathrm{C}\vec{*}) \in x$; it is total if it is cototal and the relation $\succ_1$ on x is well-founded.

Examples. **N**:

- Every total ideal is the deductive closure of a token $\mathrm{S}(\mathrm{S}\dots(\mathrm{S}0)\dots)$. The set of all tokens $\mathrm{S}(\mathrm{S}\dots(\mathrm{S}*)\dots)$ is a cototal ideal.

D (derivations):

- Total ideal $\sim$ finite derivation.
- Cototal ideal $\sim$ finite or infinite "locally correct" derivation [Mints 78].
- Arbitrary ideal $\sim$ incomplete derivation, with "holes".

Total and cototal ideals of base type

An ideal x^ι is cototal if every constructor tree $P(*) \in x$ has a "$\succ_1$-predecessor" $P(\mathrm{C}\vec{*}) \in x$; it is total if it is cototal and the relation $\succ_1$ on x is well-founded.

Examples. **N**:

- Every total ideal is the deductive closure of a token $\mathrm{S}(\mathrm{S}\dots(\mathrm{S}0)\dots)$. The set of all tokens $\mathrm{S}(\mathrm{S}\dots(\mathrm{S}*)\dots)$ is a cototal ideal.

D (derivations):

- Total ideal $\sim$ finite derivation.
- Cototal ideal $\sim$ finite or infinite "locally correct" derivation [Mints 78].
- Arbitrary ideal $\sim$ incomplete derivation, with "holes".

Total and cototal ideals of base type

An ideal x^ι is cot
otal if every constructor tree $P(*) \in x$ has a "$\succ_1$-predecessor" $P(\mathrm{C}\vec{*}) \in x$; it is total if it is cototal and the relation $\succ_1$ on x is well-founded.

Examples. **N**:

- Every total ideal is the deductive closure of a token $\mathrm{S}(\mathrm{S}\dots(\mathrm{S}0)\dots)$. The set of all tokens $\mathrm{S}(\mathrm{S}\dots(\mathrm{S}*)\dots)$ is a cototal ideal.

D (derivations):

- Total ideal $\sim$ finite derivation.
- Cototal ideal $\sim$ finite or infinite "locally correct" derivation [Mints 78].
- Arbitrary ideal $\sim$ incomplete derivation, with "holes".

Tokens, consistency and entailment at function types

Ideals: partial continuous functionals $f^{\rho\to\sigma}$ (Scott, Ershov).

- Tokens of type $\rho \to \sigma$ are pairs (U, a) with $U \in \mathrm{Con}_\rho$.
- $\{ (U_i, a_i) \mid i \in I \} \in \mathrm{Con}_{\rho\to\sigma}$ means

$$\forall_{J\subseteq I}(\bigcup_{j\in J} U_j \in \mathrm{Con}_\rho \to \{ a_j \mid j \in J \} \in \mathrm{Con}_\sigma).$$

  "Formal neighborhood".
- $W \vdash_{\rho\to\sigma} (U, a)$ means $WU \vdash_\sigma a$, where application WU of $W = \{ (U_i, a_i) \mid i \in I \}$ to U is $\{ a_i \mid U \vdash_\rho U_i \}$.

Application of $f^{\rho\to\sigma}$ to x^ρ is

$$f(x) := \{ a^\sigma \mid \exists_{U\subseteq x}(U, a) \in f \}.$$

Principles of finite support and monotonicity hold.

Tokens, consistency and entailment at function types

Ideals: partial continuous functionals $f^{\rho\to\sigma}$ (Scott, Ershov).

- Tokens of type $\rho \to \sigma$ are pairs (U, a) with $U \in \mathrm{Con}_\rho$.
- $\{ (U_i, a_i) \mid i \in I \} \in \mathrm{Con}_{\rho\to\sigma}$ means

  $$\forall_{J\subseteq I}(\bigcup_{j\in J} U_j \in \mathrm{Con}_\rho \to \{ a_j \mid j \in J \} \in \mathrm{Con}_\sigma).$$

  "Formal neighborhood".
- $W \vdash_{\rho\to\sigma} (U, a)$ means $WU \vdash_\sigma a$, where application WU of $W = \{ (U_i, a_i) \mid i \in I \}$ to U is $\{ a_i \mid U \vdash_\rho U_i \}$.

Application of $f^{\rho\to\sigma}$ to x^ρ is

$$f(x) := \{ a^\sigma \mid \exists_{U\subseteq x}(U, a) \in f \}.$$

Principles of finite support and monotonicity hold.

Tokens, consistency and entailment at function types

Ideals: partial continuous functionals $f^{\rho \to \sigma}$ (Scott, Ershov).

- Tokens of type $\rho \to \sigma$ are pairs (U, a) with $U \in \mathrm{Con}_\rho$.
- $\{ (U_i, a_i) \mid i \in I \} \in \mathrm{Con}_{\rho \to \sigma}$ means

  $$\forall_{J \subseteq I} (\bigcup_{j \in J} U_j \in \mathrm{Con}_\rho \to \{ a_j \mid j \in J \} \in \mathrm{Con}_\sigma).$$

  "Formal neighborhood".
- $W \vdash_{\rho \to \sigma} (U, a)$ means $WU \vdash_\sigma a$, where application WU of $W = \{ (U_i, a_i) \mid i \in I \}$ to U is $\{ a_i \mid U \vdash_\rho U_i \}$.

Application of $f^{\rho \to \sigma}$ to x^ρ is

$$f(x) := \{ a^\sigma \mid \exists_{U \subseteq x} (U, a) \in f \}.$$

Principles of finite support and monotonicity hold.

Tokens, consistency and entailment at function types

Ideals: partial continuous functionals $f^{\rho \to \sigma}$ (Scott, Ershov).

- Tokens of type $\rho \to \sigma$ are pairs (U, a) with $U \in \mathrm{Con}_\rho$.
- $\{ (U_i, a_i) \mid i \in I \} \in \mathrm{Con}_{\rho \to \sigma}$ means

$$\forall_{J \subseteq I} (\bigcup_{j \in J} U_j \in \mathrm{Con}_\rho \to \{ a_j \mid j \in J \} \in \mathrm{Con}_\sigma).$$

  "Formal neighborhood".

- $W \vdash_{\rho \to \sigma} (U, a)$ means $WU \vdash_\sigma a$, where application WU of $W = \{ (U_i, a_i) \mid i \in I \}$ to U is $\{ a_i \mid U \vdash_\rho U_i \}$.

Application of $f^{\rho \to \sigma}$ to x^ρ is

$$f(x) := \{ a^\sigma \mid \exists_{U \subseteq x} (U, a) \in f \}.$$

Principles of finite support and monotonicity hold.

Tokens, consistency and entailment at function types

Ideals: partial continuous functionals $f^{\rho\to\sigma}$ (Scott, Ershov).

- Tokens of type $\rho \to \sigma$ are pairs (U, a) with $U \in \mathrm{Con}_\rho$.
- $\{\, (U_i, a_i) \mid i \in I \,\} \in \mathrm{Con}_{\rho\to\sigma}$ means

  $$\forall_{J\subseteq I}(\textstyle\bigcup_{j\in J} U_j \in \mathrm{Con}_\rho \to \{\, a_j \mid j \in J \,\} \in \mathrm{Con}_\sigma).$$

  "Formal neighborhood".
- $W \vdash_{\rho\to\sigma} (U, a)$ means $WU \vdash_\sigma a$, where application WU of $W = \{\, (U_i, a_i) \mid i \in I \,\}$ to U is $\{\, a_i \mid U \vdash_\rho U_i \,\}$.

Application of $f^{\rho\to\sigma}$ to x^ρ is

$$f(x) := \{\, a^\sigma \mid \exists_{U\subseteq x}(U, a) \in f \,\}.$$

Principles of finite support and monotonicity hold.

Tokens, consistency and entailment at function types

Ideals: partial continuous functionals $f^{\rho \to \sigma}$ (Scott, Ershov).

- Tokens of type $\rho \to \sigma$ are pairs (U, a) with $U \in \mathrm{Con}_\rho$.
- $\{ (U_i, a_i) \mid i \in I \} \in \mathrm{Con}_{\rho \to \sigma}$ means

  $$\forall_{J \subseteq I} (\bigcup_{j \in J} U_j \in \mathrm{Con}_\rho \to \{ a_j \mid j \in J \} \in \mathrm{Con}_\sigma).$$

  "Formal neighborhood".
- $W \vdash_{\rho \to \sigma} (U, a)$ means $WU \vdash_\sigma a$, where application WU of $W = \{ (U_i, a_i) \mid i \in I \}$ to U is $\{ a_i \mid U \vdash_\rho U_i \}$.

Application of $f^{\rho \to \sigma}$ to x^ρ is

$$f(x) := \{ a^\sigma \mid \exists_{U \subseteq x} (U, a) \in f \}.$$

Principles of finite support and monotonicity hold.

Tokens, consistency and entailment at function types

Ideals: partial continuous functionals $f^{\rho \to \sigma}$ (Scott, Ershov).

- Tokens of type $\rho \to \sigma$ are pairs (U, a) with $U \in \mathrm{Con}_\rho$.
- $\{ (U_i, a_i) \mid i \in I \} \in \mathrm{Con}_{\rho \to \sigma}$ means

$$\forall_{J \subseteq I} (\bigcup_{j \in J} U_j \in \mathrm{Con}_\rho \to \{ a_j \mid j \in J \} \in \mathrm{Con}_\sigma).$$

  "Formal neighborhood".
- $W \vdash_{\rho \to \sigma} (U, a)$ means $WU \vdash_\sigma a$, where application WU of $W = \{ (U_i, a_i) \mid i \in I \}$ to U is $\{ a_i \mid U \vdash_\rho U_i \}$.

Application of $f^{\rho \to \sigma}$ to x^ρ is

$$f(x) := \{ a^\sigma \mid \exists_{U \subseteq x} (U, a) \in f \}.$$

Principles of finite support and monotonicity hold.

Computable functionals

A partial continuous functional f^ρ is computable if it is a (primitive) recursively enumerable set of tokens.

How to define computable functionals? By computation rules

$$D\vec{P}_i(\vec{y}_i) = M_i \qquad (i = 1, \dots, n)$$

with free variables of $\vec{P}_i(\vec{y}_i)$ and M_i among $\vec{y}_i$, where $\vec{P}_i(\vec{y}_i)$ are "constructor patterns".

Terms (a common extension of Gödel's T and Plotkin's PCF)

$$M, N ::= x^\rho \mid \mathrm{C}^\rho \mid D^\rho \mid (\lambda_{x^\rho} M^\sigma)^{\rho\to\sigma} \mid (M^{\rho\to\sigma} N^\rho)^\sigma.$$

Computable functionals

A partial continuous functional f^ρ is computable if it is a (primitive) recursively enumerable set of tokens.

How to define computable functionals? By computation rules

$$D\vec{P}_i(\vec{y}_i) = M_i \qquad (i = 1, \dots, n)$$

with free variables of $\vec{P}_i(\vec{y}_i)$ and M_i among $\vec{y}_i$, where $\vec{P}_i(\vec{y}_i)$ are "constructor patterns".

Terms (a common extension of Gödel's T and Plotkin's PCF)

$$M, N ::= x^\rho \mid \mathrm{C}^\rho \mid D^\rho \mid (\lambda_{x^\rho} M^\sigma)^{\rho\to\sigma} \mid (M^{\rho\to\sigma} N^\rho)^\sigma.$$

Computable functionals

A partial continuous functional f^ρ is computable if it is a (primitive) recursively enumerable set of tokens.

How to define computable functionals? By computation rules

$$D\vec{P}_i(\vec{y}_i) = M_i \qquad (i = 1, \dots, n)$$

with free variables of $\vec{P}_i(\vec{y}_i)$ and M_i among $\vec{y}_i$, where $\vec{P}_i(\vec{y}_i)$ are "constructor patterns".

Terms (a common extension of Gödel's T and Plotkin's PCF)

$$M, N ::= x^\rho \mid \mathrm{C}^\rho \mid D^\rho \mid (\lambda_{x^\rho} M^\sigma)^{\rho\to\sigma} \mid (M^{\rho\to\sigma} N^\rho)^\sigma.$$

Computable functionals

A partial continuous functional f^ρ is computable if it is a (primitive) recursively enumerable set of tokens.

How to define computable functionals? By computation rules

$$D\vec{P}_i(\vec{y}_i) = M_i \qquad (i = 1, \dots, n)$$

with free variables of $\vec{P}_i(\vec{y}_i)$ and M_i among $\vec{y}_i$, where $\vec{P}_i(\vec{y}_i)$ are "constructor patterns".

Terms (a common extension of Gödel's T and Plotkin's PCF)

$$M, N ::= x^\rho \mid \mathrm{C}^\rho \mid D^\rho \mid (\lambda_{x^\rho} M^\sigma)^{\rho\to\sigma} \mid (M^{\rho\to\sigma} N^\rho)^\sigma.$$

Computable functionals

A partial continuous functional f^ρ is computable if it is a (primitive) recursively enumerable set of tokens.

How to define computable functionals? By computation rules

$$D\vec{P}_i(\vec{y}_i) = M_i \qquad (i = 1, \dots, n)$$

with free variables of $\vec{P}_i(\vec{y}_i)$ and M_i among $\vec{y}_i$, where $\vec{P}_i(\vec{y}_i)$ are "constructor patterns".

Terms (a common extension of Gödel's T and Plotkin's PCF)

$$M, N ::= x^\rho \mid \mathrm{C}^\rho \mid D^\rho \mid (\lambda_{x^\rho} M^\sigma)^{\rho\to\sigma} \mid (M^{\rho\to\sigma} N^\rho)^\sigma.$$

Examples

$+ : \mathbf{N} \to \mathbf{N} \to \mathbf{N}$ defined by

$$n + 0 = n,$$
$$n + \mathrm{S}m = \mathrm{S}(n + m).$$

$Y : (\tau \to \tau) \to \tau$ defined by

$$Yf = f(Yf).$$

$\mathcal{R}^{\tau}_{\mathbf{N}} : \mathbf{N} \to \tau \to (\mathbf{N} \to \tau \to \tau) \to \tau$ defined by

$$\mathcal{R}^{\tau}_{\mathbf{N}} 0xf = x,$$
$$\mathcal{R}^{\tau}_{\mathbf{N}}(\mathrm{S}n)xf = fx(\mathcal{R}^{\tau}_{\mathbf{N}} nxf).$$

Reduction (including β, η) is non-terminating, but confluent.

Examples

$+ \colon \mathbf{N} \to \mathbf{N} \to \mathbf{N}$ defined by

$$n + 0 = n,$$
$$n + \mathrm{S}m = \mathrm{S}(n + m).$$

$Y \colon (\tau \to \tau) \to \tau$ defined by

$$Yf = f(Yf).$$

$\mathcal{R}_{\mathbf{N}}^{\tau} \colon \mathbf{N} \to \tau \to (\mathbf{N} \to \tau \to \tau) \to \tau$ defined by

$$\mathcal{R}_{\mathbf{N}}^{\tau} 0xf = x,$$
$$\mathcal{R}_{\mathbf{N}}^{\tau}(\mathrm{S}n)xf = fx(\mathcal{R}_{\mathbf{N}}^{\tau} nxf).$$

Reduction (including β, η) is non-terminating, but confluent.

Examples

$+ \colon \mathbf{N} \to \mathbf{N} \to \mathbf{N}$ defined by

$$n + 0 = n,$$
$$n + \mathrm{S}m = \mathrm{S}(n + m).$$

$Y \colon (\tau \to \tau) \to \tau$ defined by

$$Yf = f(Yf).$$

$\mathcal{R}^{\tau}_{\mathbf{N}} \colon \mathbf{N} \to \tau \to (\mathbf{N} \to \tau \to \tau) \to \tau$ defined by

$$\mathcal{R}^{\tau}_{\mathbf{N}} 0xf = x,$$
$$\mathcal{R}^{\tau}_{\mathbf{N}} (\mathrm{S}n)xf = fx(\mathcal{R}^{\tau}_{\mathbf{N}} nxf).$$

Reduction (including β, η) is non-terminating, but confluent.

Examples

$+ \colon \mathbf{N} \to \mathbf{N} \to \mathbf{N}$ defined by

$$n + 0 = n,$$
$$n + \mathrm{S}m = \mathrm{S}(n + m).$$

$Y \colon (\tau \to \tau) \to \tau$ defined by

$$Yf = f(Yf).$$

$\mathcal{R}^{\tau}_{\mathbf{N}} \colon \mathbf{N} \to \tau \to (\mathbf{N} \to \tau \to \tau) \to \tau$ defined by

$$\mathcal{R}^{\tau}_{\mathbf{N}} 0xf = x,$$
$$\mathcal{R}^{\tau}_{\mathbf{N}} (\mathrm{S}n)xf = fx(\mathcal{R}^{\tau}_{\mathbf{N}} nxf).$$

Reduction (including β, η) is non-terminating, but confluent.

Denotational semantics

How to use computation rules to define a computable functional?
Inductively define $(\vec{U}, a) \in [\![\lambda_{\vec{x}} M]\!]$ $(\mathrm{FV}(M) \subseteq \{\vec{x}\})$.
Case $\lambda_{\vec{x},y,\vec{z}} M$ with $\vec{x}$ free in M, but not y.

$$\frac{(\vec{U}, \vec{W}, a) \in [\![\lambda_{\vec{x},\vec{z}} M]\!]}{(\vec{U}, V, \vec{W}, a) \in [\![\lambda_{\vec{x},y,\vec{z}} M]\!]} (K).$$

Case $\lambda_{\vec{x}} M$ with $\vec{x}$ the free variables in M.

$$\frac{U \vdash a}{(U, a) \in [\![\lambda_x x]\!]} (V), \quad \frac{(\vec{U}, V, a) \in [\![\lambda_{\vec{x}} M]\!] \quad (\vec{U}, V) \subseteq [\![\lambda_{\vec{x}} N]\!]}{(\vec{U}, a) \in [\![\lambda_{\vec{x}} (MN)]\!]} (A).$$

For every constructor C and defined constant D:

$$\frac{\vec{U} \vdash \vec{a^*}}{(\vec{U}, \mathrm{C}\vec{a^*}) \in [\![\mathrm{C}]\!]} (\mathrm{C}), \quad \frac{(\vec{V}, a) \in [\![\lambda_{\vec{x}} M]\!] \quad \vec{U} \vdash \vec{P}(\vec{V})}{(\vec{U}, a) \in [\![D]\!]} (D),$$

with one rule (D) for every defining equation $D\vec{P}(\vec{x}) = M$.

Denotational semantics

How to use computation rules to define a computable functional?

Inductively define $(\vec{U}, a) \in [\![\lambda_{\vec{x}} M]\!]$ $(\mathrm{FV}(M) \subseteq \{\vec{x}\})$.

Case $\lambda_{\vec{x},y,\vec{z}} M$ with $\vec{x}$ free in M, but not y.

$$\frac{(\vec{U}, \vec{W}, a) \in [\![\lambda_{\vec{x},\vec{z}} M]\!]}{(\vec{U}, V, \vec{W}, a) \in [\![\lambda_{\vec{x},y,\vec{z}} M]\!]}(K).$$

Case $\lambda_{\vec{x}} M$ with $\vec{x}$ the free variables in M.

$$\frac{U \vdash a}{(U, a) \in [\![\lambda_x x]\!]}(V), \quad \frac{(\vec{U}, V, a) \in [\![\lambda_{\vec{x}} M]\!] \quad (\vec{U}, V) \subseteq [\![\lambda_{\vec{x}} N]\!]}{(\vec{U}, a) \in [\![\lambda_{\vec{x}}(MN)]\!]}(A).$$

For every constructor C and defined constant D:

$$\frac{\vec{U} \vdash \vec{a^*}}{(\vec{U}, \mathrm{C}\vec{a^*}) \in [\![\mathrm{C}]\!]}(\mathrm{C}), \quad \frac{(\vec{V}, a) \in [\![\lambda_{\vec{x}} M]\!] \quad \vec{U} \vdash \vec{P}(\vec{V})}{(\vec{U}, a) \in [\![D]\!]}(D),$$

with one rule (D) for every defining equation $D\vec{P}(\vec{x}) = M$.

Denotational semantics

How to use computation rules to define a computable functional?
Inductively define $(\vec{U}, a) \in [\![\lambda_{\vec{x}} M]\!]$ $(\mathrm{FV}(M) \subseteq \{\vec{x}\,\})$.

Case $\lambda_{\vec{x},y,\vec{z}} M$ with $\vec{x}$ free in M, but not y.

$$\frac{(\vec{U}, \vec{W}, a) \in [\![\lambda_{\vec{x},\vec{z}} M]\!]}{(\vec{U}, V, \vec{W}, a) \in [\![\lambda_{\vec{x},y,\vec{z}} M]\!]}(K).$$

Case $\lambda_{\vec{x}} M$ with $\vec{x}$ the free variables in M.

$$\frac{U \vdash a}{(U, a) \in [\![\lambda_x x]\!]}(V), \quad \frac{(\vec{U}, V, a) \in [\![\lambda_{\vec{x}} M]\!] \quad (\vec{U}, V) \subseteq [\![\lambda_{\vec{x}} N]\!]}{(\vec{U}, a) \in [\![\lambda_{\vec{x}}(MN)]\!]}(A).$$

For every constructor C and defined constant D:

$$\frac{\vec{U} \vdash \vec{a^*}}{(\vec{U}, \mathrm{C}\vec{a^*}) \in [\![\mathrm{C}]\!]}(\mathrm{C}), \quad \frac{(\vec{V}, a) \in [\![\lambda_{\vec{x}} M]\!] \quad \vec{U} \vdash \vec{P}(\vec{V})}{(\vec{U}, a) \in [\![D]\!]}(D),$$

with one rule (D) for every defining equation $D\vec{P}(\vec{x}\,) = M$.

Denotational semantics

How to use computation rules to define a computable functional?
Inductively define $(\vec{U}, a) \in [\![\lambda_{\vec{x}} M]\!]$ $(\mathrm{FV}(M) \subseteq \{\vec{x}\})$.
Case $\lambda_{\vec{x},y,\vec{z}} M$ with $\vec{x}$ free in M, but not y.

$$\frac{(\vec{U}, \vec{W}, a) \in [\![\lambda_{\vec{x},\vec{z}} M]\!]}{(\vec{U}, V, \vec{W}, a) \in [\![\lambda_{\vec{x},y,\vec{z}} M]\!]}(K).$$

Case $\lambda_{\vec{x}} M$ with $\vec{x}$ the free variables in M.

$$\frac{U \vdash a}{(U, a) \in [\![\lambda_x x]\!]}(V), \quad \frac{(\vec{U}, V, a) \in [\![\lambda_{\vec{x}} M]\!] \quad (\vec{U}, V) \subseteq [\![\lambda_{\vec{x}} N]\!]}{(\vec{U}, a) \in [\![\lambda_{\vec{x}}(MN)]\!]}(A).$$

For every constructor C and defined constant D:

$$\frac{\vec{U} \vdash \vec{a^*}}{(\vec{U}, \mathrm{C}\vec{a^*}) \in [\![\mathrm{C}]\!]}(\mathrm{C}), \quad \frac{(\vec{V}, a) \in [\![\lambda_{\vec{x}} M]\!] \quad \vec{U} \vdash \vec{P}(\vec{V})}{(\vec{U}, a) \in [\![D]\!]}(D),$$

with one rule (D) for every defining equation $D\vec{P}(\vec{x}) = M$.

Denotational semantics

How to use computation rules to define a computable functional?
Inductively define $(\vec{U}, a) \in [\![\lambda_{\vec{x}} M]\!]$ $(\mathrm{FV}(M) \subseteq \{\vec{x}\})$.
Case $\lambda_{\vec{x},y,\vec{z}} M$ with $\vec{x}$ free in M, but not y.

$$\frac{(\vec{U}, \vec{W}, a) \in [\![\lambda_{\vec{x},\vec{z}} M]\!]}{(\vec{U}, V, \vec{W}, a) \in [\![\lambda_{\vec{x},y,\vec{z}} M]\!]}(K).$$

Case $\lambda_{\vec{x}} M$ with $\vec{x}$ the free variables in M.

$$\frac{U \vdash a}{(U, a) \in [\![\lambda_x x]\!]}(V), \quad \frac{(\vec{U}, V, a) \in [\![\lambda_{\vec{x}} M]\!] \quad (\vec{U}, V) \subseteq [\![\lambda_{\vec{x}} N]\!]}{(\vec{U}, a) \in [\![\lambda_{\vec{x}}(MN)]\!]}(A).$$

For every constructor C and defined constant D:

$$\frac{\vec{U} \vdash \vec{a^*}}{(\vec{U}, \mathrm{C}\vec{a^*}) \in [\![\mathrm{C}]\!]}(\mathrm{C}), \quad \frac{(\vec{V}, a) \in [\![\lambda_{\vec{x}} M]\!] \quad \vec{U} \vdash \vec{P}(\vec{V})}{(\vec{U}, a) \in [\![D]\!]}(D),$$

with one rule (D) for every defining equation $D\vec{P}(\vec{x}) = M$.

Denotational semantics

How to use computation rules to define a computable functional?
Inductively define $(\vec{U}, a) \in [\![\lambda_{\vec{x}} M]\!]$ $(\mathrm{FV}(M) \subseteq \{\vec{x}\})$.
Case $\lambda_{\vec{x},y,\vec{z}} M$ with $\vec{x}$ free in M, but not y.

$$\frac{(\vec{U}, \vec{W}, a) \in [\![\lambda_{\vec{x},\vec{z}} M]\!]}{(\vec{U}, V, \vec{W}, a) \in [\![\lambda_{\vec{x},y,\vec{z}} M]\!]}(K).$$

Case $\lambda_{\vec{x}} M$ with $\vec{x}$ the free variables in M.

$$\frac{U \vdash a}{(U, a) \in [\![\lambda_x x]\!]}(V), \quad \frac{(\vec{U}, V, a) \in [\![\lambda_{\vec{x}} M]\!] \quad (\vec{U}, V) \subseteq [\![\lambda_{\vec{x}} N]\!]}{(\vec{U}, a) \in [\![\lambda_{\vec{x}}(MN)]\!]}(A).$$

For every constructor C and defined constant D:

$$\frac{\vec{U} \vdash \vec{a^*}}{(\vec{U}, \mathrm{C}\vec{a^*}) \in [\![\mathrm{C}]\!]}(\mathrm{C}), \quad \frac{(\vec{V}, a) \in [\![\lambda_{\vec{x}} M]\!] \quad \vec{U} \vdash \vec{P}(\vec{V})}{(\vec{U}, a) \in [\![D]\!]}(D),$$

with one rule (D) for every defining equation $D\vec{P}(\vec{x}) = M$.

Properties of the denotational semantics

- The value is preserved under standard β, η-conversion and the computation rules.
- An adequacy theorem holds: whenever a closed term M^ι has a proper token in its denotation $[\![M]\!]$, then M (head) reduces to a constructor term entailing this token.

Properties of the denotational semantics

- The value is preserved under standard β, η-conversion and the computation rules.
- An adequacy theorem holds: whenever a closed term M^ι has a proper token in its denotation $[\![M]\!]$, then M (head) reduces to a constructor term entailing this token.

Properties of the denotational semantics

- The value is preserved under standard β, η-conversion and the computation rules.
- An adequacy theorem holds: whenever a closed term M^ι has a proper token in its denotation $[\![M]\!]$, then M (head) reduces to a constructor term entailing this token.

A theory of computable functionals (TCF)

A variant of HA^ω.

Formulas A and predicates P are defined simultaneously

$$A, B ::= P\vec{r} \mid A \to B \mid \forall_x A$$
$$P ::= X \mid \{\, \vec{x} \mid A \,\} \mid I \qquad (I \text{ inductively defined}).$$

$\forall_X A$ not allowed, since this would be impredicative: in the predicate existence axiom $P := \{\, \vec{x} \mid A \,\}$ the formula A could contain quantifiers with the newly created P in its range.

$\forall_{x^\rho} A$ is unproblematic: no such existence axioms.

A theory of computable functionals (TCF)

A variant of HA^ω.

Formulas A and predicates P are defined simultaneously

$$A, B ::= P\vec{r} \mid A \to B \mid \forall_x A$$
$$P ::= X \mid \{\, \vec{x} \mid A \,\} \mid I \qquad (I \text{ inductively defined}).$$

$\forall_X A$ not allowed, since this would be impredicative: in the predicate existence axiom $P := \{\, \vec{x} \mid A \,\}$ the formula A could contain quantifiers with the newly created P in its range.

$\forall_{x^\rho} A$ is unproblematic: no such existence axioms.

A theory of computable functionals (TCF)

A variant of HA^ω.

Formulas A and predicates P are defined simultaneously

$$A, B ::= P\vec{r} \mid A \to B \mid \forall_x A$$
$$P ::= X \mid \{\, \vec{x} \mid A \,\} \mid I \qquad (I \text{ inductively defined}).$$

$\forall_X A$ not allowed, since this would be impredicative: in the predicate existence axiom $P := \{\, \vec{x} \mid A \,\}$ the formula A could contain quantifiers with the newly created P in its range.

$\forall_{x^\rho} A$ is unproblematic: no such existence axioms.

A theory of computable functionals (TCF)

A variant of HA^ω.

Formulas A and predicates P are defined simultaneously

$$A, B ::= P\vec{r} \mid A \to B \mid \forall_x A$$
$$P ::= X \mid \{\, \vec{x} \mid A \,\} \mid I \qquad (I \text{ inductively defined}).$$

$\forall_X A$ not allowed, since this would be impredicative: in the predicate existence axiom $P := \{\, \vec{x} \mid A \,\}$ the formula A could contain quantifiers with the newly created P in its range.

$\forall_{x^\rho} A$ is unproblematic: no such existence axioms.

A theory of computable functionals (TCF)

A variant of HA^ω.

Formulas A and predicates P are defined simultaneously

$$A, B ::= P\vec{r} \mid A \to B \mid \forall_x A$$
$$P ::= X \mid \{\, \vec{x} \mid A \,\} \mid I \qquad (I \text{ inductively defined}).$$

$\forall_X A$ not allowed, since this would be impredicative: in the predicate existence axiom $P := \{\, \vec{x} \mid A \,\}$ the formula A could contain quantifiers with the newly created P in its range.

$\forall_{x^\rho} A$ is unproblematic: no such existence axioms.

Brouwer - Heyting - Kolmogorov

Have $\to^{\pm}$, $\forall^{\pm}$, $I^{\pm}$. BHK-interpretation:

- p proves $A \to B$ iff p is a construction transforming any proof q of A into a proof $p(q)$ of B.
- p proves $\forall_{x^\rho} A(x)$ iff p is a construction such that for all a^ρ, $p(a)$ proves $A(a)$.

Leaves open:

- What is a "construction"?
- What is a proof of a prime formula?

Proposal:

- Construction: computable functional.
- Proof of a prime formula $I\vec{r}$: generation tree.

Example: generation tree for $\mathrm{Even}(6)$ should consist of a single branch with nodes $\mathrm{Even}(0)$, $\mathrm{Even}(2)$, $\mathrm{Even}(4)$ and $\mathrm{Even}(6)$.

Brouwer - Heyting - Kolmogorov

Have $\to^{\pm}$, $\forall^{\pm}$, $I^{\pm}$. BHK-interpretation:

- p proves $A \to B$ iff p is a construction transforming any proof q of A into a proof $p(q)$ of B.
- p proves $\forall_{x^\rho} A(x)$ iff p is a construction such that for all a^ρ, $p(a)$ proves $A(a)$.

Leaves open:

- What is a "construction"?
- What is a proof of a prime formula?

Proposal:

- Construction: computable functional.
- Proof of a prime formula $I\vec{r}$: generation tree.

Example: generation tree for $\mathrm{Even}(6)$ should consist of a single branch with nodes $\mathrm{Even}(0)$, $\mathrm{Even}(2)$, $\mathrm{Even}(4)$ and $\mathrm{Even}(6)$.

Brouwer - Heyting - Kolmogorov

Have $\to^{\pm}$, $\forall^{\pm}$, $I^{\pm}$. BHK-interpretation:

- p proves $A \to B$ iff p is a construction transforming any proof q of A into a proof $p(q)$ of B.
- p proves $\forall_{x^\rho} A(x)$ iff p is a construction such that for all a^ρ, $p(a)$ proves $A(a)$.

Leaves open:

- What is a "construction"?
- What is a proof of a prime formula?

Proposal:

- Construction: computable functional.
- Proof of a prime formula $I\vec{r}$: generation tree.

Example: generation tree for $\mathrm{Even}(6)$ should consist of a single branch with nodes $\mathrm{Even}(0)$, $\mathrm{Even}(2)$, $\mathrm{Even}(4)$ and $\mathrm{Even}(6)$.

Brouwer - Heyting - Kolmogorov

Have $\to^{\pm}$, $\forall^{\pm}$, $I^{\pm}$. BHK-interpretation:

- p proves $A \to B$ iff p is a construction transforming any proof q of A into a proof $p(q)$ of B.
- p proves $\forall_{x^\rho} A(x)$ iff p is a construction such that for all a^ρ, $p(a)$ proves $A(a)$.

Leaves open:

- What is a "construction"?
- What is a proof of a prime formula?

Proposal:

- Construction: computable functional.
- Proof of a prime formula $I\vec{r}$: generation tree.

Example: generation tree for $\mathrm{Even}(6)$ should consist of a single branch with nodes $\mathrm{Even}(0)$, $\mathrm{Even}(2)$, $\mathrm{Even}(4)$ and $\mathrm{Even}(6)$.

Brouwer - Heyting - Kolmogorov

Have $\to^{\pm}$, $\forall^{\pm}$, $I^{\pm}$. BHK-interpretation:

- p proves $A \to B$ iff p is a construction transforming any proof q of A into a proof $p(q)$ of B.
- p proves $\forall_{x^\rho} A(x)$ iff p is a construction such that for all a^ρ, $p(a)$ proves $A(a)$.

Leaves open:

- What is a "construction"?
- What is a proof of a prime formula?

Proposal:

- Construction: computable functional.
- Proof of a prime formula $I\vec{r}$: generation tree.

Example: generation tree for $\mathrm{Even}(6)$ should consist of a single branch with nodes $\mathrm{Even}(0)$, $\mathrm{Even}(2)$, $\mathrm{Even}(4)$ and $\mathrm{Even}(6)$.

Brouwer - Heyting - Kolmogorov

Have $\to^{\pm}$, $\forall^{\pm}$, $I^{\pm}$. BHK-interpretation:

- p proves $A \to B$ iff p is a construction transforming any proof q of A into a proof $p(q)$ of B.
- p proves $\forall_{x^\rho} A(x)$ iff p is a construction such that for all a^ρ, $p(a)$ proves $A(a)$.

Leaves open:

- What is a "construction"?
- What is a proof of a prime formula?

Proposal:

- Construction: computable functional.
- Proof of a prime formula $I\vec{r}$: generation tree.

Example: generation tree for $\mathrm{Even}(6)$ should consist of a single branch with nodes $\mathrm{Even}(0)$, $\mathrm{Even}(2)$, $\mathrm{Even}(4)$ and $\mathrm{Even}(6)$.

Brouwer - Heyting - Kolmogorov

Have $\to^{\pm}$, $\forall^{\pm}$, $I^{\pm}$. BHK-interpretation:

- p proves $A \to B$ iff p is a construction transforming any proof q of A into a proof $p(q)$ of B.
- p proves $\forall_{x^\rho} A(x)$ iff p is a construction such that for all a^ρ, $p(a)$ proves $A(a)$.

Leaves open:

- What is a "construction"?
- What is a proof of a prime formula?

Proposal:

- Construction: computable functional.
- Proof of a prime formula $I\vec{r}$: generation tree.

Example: generation tree for $\mathrm{Even}(6)$ should consist of a single branch with nodes $\mathrm{Even}(0)$, $\mathrm{Even}(2)$, $\mathrm{Even}(4)$ and $\mathrm{Even}(6)$.

The type $\tau(A)$ of a formula A

Distinguish non-computational (n.c.) (or Harrop) and computationally relevant (c.r.) formulas. Example:

- $r = s$ is n.c.
- $\mathrm{Even}(n)$ is c.r.

Extend the use of $\rho \to \sigma$ to the "nulltype symbol" $\circ$:

$$(\rho \to \circ) := \circ, \quad (\circ \to \sigma) := \sigma, \quad (\circ \to \circ) := \circ.$$

Define the type $\tau(A)$ of a formula A by

$$\tau(I\vec{r}\,) = \begin{cases} \iota_I & \text{if } I \text{ is c.r.,} \\ \circ & \text{if } I \text{ is n.c.,} \end{cases}$$
$$\tau(A \to B) := \tau(A) \to \tau(B),$$
$$\tau(\forall_{x^\rho} A) := \rho \to \tau(A)$$

with ι_I associated naturally with I.

The type $\tau(A)$ of a formula A

Distinguish non-computational (n.c.) (or Harrop) and computationally relevant (c.r.) formulas. Example:

- $r = s$ is n.c.
- $\mathrm{Even}(n)$ is c.r.

Extend the use of $\rho \to \sigma$ to the "nulltype symbol" $\circ$:

$$(\rho \to \circ) := \circ, \quad (\circ \to \sigma) := \sigma, \quad (\circ \to \circ) := \circ.$$

Define the type $\tau(A)$ of a formula A by

$$\tau(I\vec{r}) = \begin{cases} \iota_I & \text{if } I \text{ is c.r.,} \\ \circ & \text{if } I \text{ is n.c.,} \end{cases}$$

$$\tau(A \to B) := \tau(A) \to \tau(B),$$

$$\tau(\forall_{x^\rho} A) := \rho \to \tau(A)$$

with ι_I associated naturally with I.

The type $\tau(A)$ of a formula A

Distinguish non-computational (n.c.) (or Harrop) and computationally relevant (c.r.) formulas. Example:

- $r = s$ is n.c.
- $\mathrm{Even}(n)$ is c.r.

Extend the use of $\rho \to \sigma$ to the "nulltype symbol" $\circ$:

$$(\rho \to \circ) := \circ, \quad (\circ \to \sigma) := \sigma, \quad (\circ \to \circ) := \circ.$$

Define the type $\tau(A)$ of a formula A by

$$\tau(I\vec{r}\,) = \begin{cases} \iota_I & \text{if } I \text{ is c.r.,} \\ \circ & \text{if } I \text{ is n.c.,} \end{cases}$$
$$\tau(A \to B) := \tau(A) \to \tau(B),$$
$$\tau(\forall_{x^\rho} A) := \rho \to \tau(A)$$

with ι_I associated naturally with I.

The type $\tau(A)$ of a formula A

Distinguish non-computational (n.c.) (or Harrop) and computationally relevant (c.r.) formulas. Example:

- $r = s$ is n.c.
- $\mathrm{Even}(n)$ is c.r.

Extend the use of $\rho \to \sigma$ to the "nulltype symbol" $\circ$:

$$(\rho \to \circ) := \circ, \quad (\circ \to \sigma) := \sigma, \quad (\circ \to \circ) := \circ.$$

Define the type $\tau(A)$ of a formula A by

$$\tau(I\vec{r}) = \begin{cases} \iota_I & \text{if } I \text{ is c.r.,} \\ \circ & \text{if } I \text{ is n.c.,} \end{cases}$$
$$\tau(A \to B) := \tau(A) \to \tau(B),$$
$$\tau(\forall_{x^\rho} A) := \rho \to \tau(A)$$

with ι_I associated naturally with I.

The type $\tau(A)$ of a formula A

Distinguish non-computational (n.c.) (or Harrop) and computationally relevant (c.r.) formulas. Example:

- $r = s$ is n.c.
- $\mathrm{Even}(n)$ is c.r.

Extend the use of $\rho \to \sigma$ to the "nulltype symbol" $\circ$:

$$(\rho \to \circ) := \circ, \quad (\circ \to \sigma) := \sigma, \quad (\circ \to \circ) := \circ.$$

Define the type $\tau(A)$ of a formula A by

$$\tau(I\vec{r}\,) = \begin{cases} \iota_I & \text{if } I \text{ is c.r.,} \\ \circ & \text{if } I \text{ is n.c.,} \end{cases}$$
$$\tau(A \to B) := \tau(A) \to \tau(B),$$
$$\tau(\forall_{x^\rho} A) := \rho \to \tau(A)$$

with ι_I associated naturally with I.

Realizability

Introduce a special nullterm symbol ε to be used as a "realizer" for n.c. formulas. Extend term application to ε by

$$\varepsilon t := \varepsilon, \quad t\varepsilon := t, \quad \varepsilon\varepsilon := \varepsilon.$$

Definition (t $\mathbf{r}$ A, t realizes A)

Let A be a formula and t either a term of type $\tau(A)$ if the latter is a type, or the nullterm symbol ε for n.c. A.

$$t\ \mathbf{r}\ I\vec{s} := \begin{cases} I^{\mathbf{r}} t\vec{s} & \text{if } I \text{ is c.r. } (I^{\mathbf{r}} \text{ inductively defined}), \\ I\vec{s} & \text{if } I \text{ is n.c.}, \end{cases}$$

$$t\ \mathbf{r}\ (A \to B) := \forall_x (x\ \mathbf{r}\ A \ \to\ tx\ \mathbf{r}\ B),$$

$$t\ \mathbf{r}\ \forall_x A := \forall_x (tx\ \mathbf{r}\ A).$$

Realizability

Introduce a special nullterm symbol ε to be used as a "realizer" for n.c. formulas. Extend term application to ε by

$$\varepsilon t := \varepsilon, \quad t\varepsilon := t, \quad \varepsilon\varepsilon := \varepsilon.$$

Definition ($t \mathbf{r} A$, t realizes A)

Let A be a formula and t either a term of type $\tau(A)$ if the latter is a type, or the nullterm symbol ε for n.c. A.

$$t \mathbf{r} I\vec{s} := \begin{cases} I^{\mathbf{r}} t\vec{s} & \text{if } I \text{ is c.r. } (I^{\mathbf{r}} \text{ inductively defined}), \\ I\vec{s} & \text{if } I \text{ is n.c.,} \end{cases}$$

$$t \mathbf{r} (A \to B) := \forall_x (x \mathbf{r} A \;\to\; tx \mathbf{r} B),$$

$$t \mathbf{r} \forall_x A := \forall_x (tx \mathbf{r} A).$$

Realizability

Introduce a special nullterm symbol ε to be used as a "realizer" for n.c. formulas. Extend term application to ε by

$$\varepsilon t := \varepsilon, \quad t\varepsilon := t, \quad \varepsilon\varepsilon := \varepsilon.$$

Definition (t $\mathbf{r}$ A, t realizes A)

Let A be a formula and t either a term of type $\tau(A)$ if the latter is a type, or the nullterm symbol ε for n.c. A.

$$t \mathbin{\mathbf{r}} I\vec{s} := \begin{cases} I^{\mathbf{r}} t\vec{s} & \text{if } I \text{ is c.r. } (I^{\mathbf{r}} \text{ inductively defined}), \\ I\vec{s} & \text{if } I \text{ is n.c.}, \end{cases}$$

$$t \mathbin{\mathbf{r}} (A \to B) := \forall_x (x \mathbin{\mathbf{r}} A \;\to\; tx \mathbin{\mathbf{r}} B),$$

$$t \mathbin{\mathbf{r}} \forall_x A := \forall_x (tx \mathbin{\mathbf{r}} A).$$

Realizability

Introduce a special nullterm symbol ε to be used as a "realizer" for n.c. formulas. Extend term application to ε by

$$\varepsilon t := \varepsilon, \quad t\varepsilon := t, \quad \varepsilon\varepsilon := \varepsilon.$$

Definition ($t\ \mathbf{r}\ A$, t realizes A)

Let A be a formula and t either a term of type $\tau(A)$ if the latter is a type, or the nullterm symbol ε for n.c. A.

$$t\ \mathbf{r}\ I\vec{s} := \begin{cases} I^{\mathbf{r}} t \vec{s} & \text{if } I \text{ is c.r. } (I^{\mathbf{r}} \text{ inductively defined}), \\ I\vec{s} & \text{if } I \text{ is n.c.}, \end{cases}$$

$$t\ \mathbf{r}\ (A \to B) := \forall_x (x\ \mathbf{r}\ A \ \to\ tx\ \mathbf{r}\ B),$$

$$t\ \mathbf{r}\ \forall_x A := \forall_x (tx\ \mathbf{r}\ A).$$

Realizability

Introduce a special nullterm symbol ε to be used as a "realizer" for n.c. formulas. Extend term application to ε by

$$\varepsilon t := \varepsilon, \quad t\varepsilon := t, \quad \varepsilon\varepsilon := \varepsilon.$$

Definition ($t\ \mathbf{r}\ A$, t realizes A)

Let A be a formula and t either a term of type $\tau(A)$ if the latter is a type, or the nullterm symbol ε for n.c. A.

$$t\ \mathbf{r}\ I\vec{s} := \begin{cases} I^{\mathbf{r}} t\vec{s} & \text{if } I \text{ is c.r. } (I^{\mathbf{r}} \text{ inductively defined}), \\ I\vec{s} & \text{if } I \text{ is n.c.}, \end{cases}$$

$$t\ \mathbf{r}\ (A \to B) := \forall_x (x\ \mathbf{r}\ A \ \to\ tx\ \mathbf{r}\ B),$$

$$t\ \mathbf{r}\ \forall_x A := \forall_x (tx\ \mathbf{r}\ A).$$

Extracted terms, soundness theorem

For a derivation M of a formula A define its extracted term $\mathrm{et}(M)$, of type $\tau(A)$. For M^A with A n.c. let $\mathrm{et}(M^A) := \varepsilon$. Else

$$\begin{aligned}
\mathrm{et}(u^A) &:= x_u^{\tau(A)} \quad (x_u^{\tau(A)} \text{ uniquely associated to } u^A),\\
\mathrm{et}((\lambda_{u^A} M^B)^{A\to B}) &:= \lambda_{x_u^{\tau(A)}} \mathrm{et}(M),\\
\mathrm{et}((M^{A\to B} N^A)^B) &:= \mathrm{et}(M)\mathrm{et}(N),\\
\mathrm{et}((\lambda_{x^\rho} M^A)^{\forall_x A}) &:= \lambda_{x^\rho} \mathrm{et}(M),\\
\mathrm{et}((M^{\forall_x A(x)} r)^{A(r)}) &:= \mathrm{et}(M) r.
\end{aligned}$$

Extracted terms for the axioms: let I be c.r.

$$\mathrm{et}(I_i^+) := \mathrm{C}_i, \qquad \mathrm{et}(I^-) := \mathcal{R},$$

where both the constructor C_i and the recursion operator $\mathcal{R}$ refer to the algebra ι_I associated with I.

Soundness. Let M be a derivation of A from assumptions $u_i \colon C_i$. Then we can derive $\mathrm{et}(M) \mathbf{\ r\ } A$ from assumptions $x_{u_i} \mathbf{\ r\ } C_i$.

Extracted terms, soundness theorem

For a derivation M of a formula A define its extracted term $\mathrm{et}(M)$, of type $\tau(A)$. For M^A with A n.c. let $\mathrm{et}(M^A) := \varepsilon$. Else

$$
\begin{aligned}
&\mathrm{et}(u^A) && := x_u^{\tau(A)} \quad (x_u^{\tau(A)} \text{ uniquely associated to } u^A),\\
&\mathrm{et}((\lambda_{u^A} M^B)^{A \to B}) && := \lambda_{x_u^{\tau(A)}} \mathrm{et}(M),\\
&\mathrm{et}((M^{A \to B} N^A)^B) && := \mathrm{et}(M)\mathrm{et}(N),\\
&\mathrm{et}((\lambda_{x^\rho} M^A)^{\forall_x A}) && := \lambda_{x^\rho} \mathrm{et}(M),\\
&\mathrm{et}((M^{\forall_x A(x)} r)^{A(r)}) && := \mathrm{et}(M) r.
\end{aligned}
$$

Extracted terms for the axioms: let I be c.r.

$$\mathrm{et}(I_i^+) := \mathrm{C}_i, \qquad \mathrm{et}(I^-) := \mathcal{R},$$

where both the constructor C_i and the recursion operator $\mathcal{R}$ refer to the algebra ι_I associated with I.

Soundness. Let M be a derivation of A from assumptions $u_i : C_i$. Then we can derive $\mathrm{et}(M)\ \mathbf{r}\ A$ from assumptions $x_{u_i}\ \mathbf{r}\ C_i$.

Extracted terms, soundness theorem

For a derivation M of a formula A define its extracted term $\mathrm{et}(M)$, of type $\tau(A)$. For M^A with A n.c. let $\mathrm{et}(M^A) := \varepsilon$. Else

$$
\begin{aligned}
\mathrm{et}(u^A) &:= x_u^{\tau(A)} \quad (x_u^{\tau(A)} \text{ uniquely associated to } u^A),\\
\mathrm{et}((\lambda_{u^A} M^B)^{A\to B}) &:= \lambda_{x_u^{\tau(A)}} \mathrm{et}(M),\\
\mathrm{et}((M^{A\to B} N^A)^B) &:= \mathrm{et}(M)\mathrm{et}(N),\\
\mathrm{et}((\lambda_{x^\rho} M^A)^{\forall_x A}) &:= \lambda_{x^\rho} \mathrm{et}(M),\\
\mathrm{et}((M^{\forall_x A(x)} r)^{A(r)}) &:= \mathrm{et}(M) r.
\end{aligned}
$$

Extracted terms for the axioms: let I be c.r.

$$\mathrm{et}(I_i^+) := \mathrm{C}_i, \qquad \mathrm{et}(I^-) := \mathcal{R},$$

where both the constructor C_i and the recursion operator $\mathcal{R}$ refer to the algebra ι_I associated with I.

Soundness. Let M be a derivation of A from assumptions $u_i : C_i$. Then we can derive $\mathrm{et}(M) \mathbf{r} A$ from assumptions $x_{u_i} \mathbf{r} C_i$.

Extracted terms, soundness theorem

For a derivation M of a formula A define its extracted term $\mathrm{et}(M)$, of type $\tau(A)$. For M^A with A n.c. let $\mathrm{et}(M^A) := \varepsilon$. Else

$$
\begin{aligned}
\mathrm{et}(u^A) &:= x_u^{\tau(A)} \quad (x_u^{\tau(A)} \text{ uniquely associated to } u^A),\\
\mathrm{et}((\lambda_{u^A} M^B)^{A\to B}) &:= \lambda_{x_u^{\tau(A)}} \mathrm{et}(M),\\
\mathrm{et}((M^{A\to B} N^A)^B) &:= \mathrm{et}(M)\mathrm{et}(N),\\
\mathrm{et}((\lambda_{x^\rho} M^A)^{\forall_x A}) &:= \lambda_{x^\rho} \mathrm{et}(M),\\
\mathrm{et}((M^{\forall_x A(x)} r)^{A(r)}) &:= \mathrm{et}(M) r.
\end{aligned}
$$

Extracted terms for the axioms: let I be c.r.

$$\mathrm{et}(I_i^+) := \mathrm{C}_i, \qquad \mathrm{et}(I^-) := \mathcal{R},$$

where both the constructor C_i and the recursion operator $\mathcal{R}$ refer to the algebra ι_I associated with I.

Soundness. Let M be a derivation of A from assumptions $u_i : C_i$. Then we can derive $\mathrm{et}(M) \mathbf{r} A$ from assumptions $x_{u_i} \mathbf{r} C_i$.

Extracted terms, soundness theorem

For a derivation M of a formula A define its extracted term $\mathrm{et}(M)$, of type $\tau(A)$. For M^A with A n.c. let $\mathrm{et}(M^A) := \varepsilon$. Else

$$
\begin{aligned}
\mathrm{et}(u^A) &:= x_u^{\tau(A)} \quad (x_u^{\tau(A)} \text{ uniquely associated to } u^A),\\
\mathrm{et}((\lambda_{u^A} M^B)^{A\to B}) &:= \lambda_{x_u^{\tau(A)}} \mathrm{et}(M),\\
\mathrm{et}((M^{A\to B} N^A)^B) &:= \mathrm{et}(M)\mathrm{et}(N),\\
\mathrm{et}((\lambda_{x^\rho} M^A)^{\forall_x A}) &:= \lambda_{x^\rho} \mathrm{et}(M),\\
\mathrm{et}((M^{\forall_x A(x)} r)^{A(r)}) &:= \mathrm{et}(M) r.
\end{aligned}
$$

Extracted terms for the axioms: let I be c.r.

$$\mathrm{et}(I_i^+) := \mathrm{C}_i, \qquad \mathrm{et}(I^-) := \mathcal{R},$$

where both the constructor C_i and the recursion operator $\mathcal{R}$ refer to the algebra ι_I associated with I.

Soundness. Let M be a derivation of A from assumptions $u_i : C_i$. Then we can derive $\mathrm{et}(M)\ \mathbf{r}\ A$ from assumptions $x_{u_i}\ \mathbf{r}\ C_i$.

Extracted terms, soundness theorem

For a derivation M of a formula A define its extracted term $\mathrm{et}(M)$, of type $\tau(A)$. For M^A with A n.c. let $\mathrm{et}(M^A) := \varepsilon$. Else

$$\begin{aligned}
\mathrm{et}(u^A) &:= x_u^{\tau(A)} \quad (x_u^{\tau(A)} \text{ uniquely associated to } u^A),\\
\mathrm{et}((\lambda_{u^A} M^B)^{A \to B}) &:= \lambda_{x_u^{\tau(A)}} \mathrm{et}(M),\\
\mathrm{et}((M^{A \to B} N^A)^B) &:= \mathrm{et}(M)\mathrm{et}(N),\\
\mathrm{et}((\lambda_{x^\rho} M^A)^{\forall_x A}) &:= \lambda_{x^\rho} \mathrm{et}(M),\\
\mathrm{et}((M^{\forall_x A(x)} r)^{A(r)}) &:= \mathrm{et}(M) r.
\end{aligned}$$

Extracted terms for the axioms: let I be c.r.

$$\mathrm{et}(I_i^+) := \mathrm{C}_i, \qquad \mathrm{et}(I^-) := \mathcal{R},$$

where both the constructor C_i and the recursion operator $\mathcal{R}$ refer to the algebra ι_I associated with I.

Soundness. Let M be a derivation of A from assumptions $u_i \colon C_i$. Then we can derive $\mathrm{et}(M)$ $\mathbf{r}$ A from assumptions x_{u_i} $\mathbf{r}$ C_i.

Relation of TCF to type theory

- Main difference: partial functionals are first class citizens.
- "Logic enriched": Formulas and types kept separate.
- Minimal logic: $\to, \forall$ only. $x = y$ (Leibniz equality), $\exists$, $\vee$, $\wedge$ inductively defined (Martin-Löf).
- $\bot := (\mathrm{False} = \mathrm{True})$. Ex-falso-quodlibet: $\bot \to A$ provable.
- "Decorations" $\to^{\mathrm{nc}}$, $\forall^{\mathrm{nc}}$ (i) allow abstract theory (ii) remove unused data.

Relation of TCF to type theory

- Main difference: partial functionals are first class citizens.
- "Logic enriched": Formulas and types kept separate.
- Minimal logic: $\to, \forall$ only. $x = y$ (Leibniz equality), $\exists$, $\vee$, $\wedge$ inductively defined (Martin-Löf).
- $\bot := (\mathrm{False} = \mathrm{True})$. Ex-falso-quodlibet: $\bot \to A$ provable.
- "Decorations" $\to^{\mathrm{nc}}$, $\forall^{\mathrm{nc}}$ (i) allow abstract theory (ii) remove unused data.

Relation of TCF to type theory

- Main difference: partial functionals are first class citizens.
- "Logic enriched": Formulas and types kept separate.
- Minimal logic: $\rightarrow, \forall$ only. $x = y$ (Leibniz equality), $\exists$, $\vee$, $\wedge$ inductively defined (Martin-Löf).
- $\bot := (\mathrm{False} = \mathrm{True})$. Ex-falso-quodlibet: $\bot \rightarrow A$ provable.
- "Decorations" $\rightarrow^{\mathrm{nc}}$, $\forall^{\mathrm{nc}}$ (i) allow abstract theory (ii) remove unused data.

Relation of TCF to type theory

- Main difference: partial functionals are first class citizens.
- "Logic enriched": Formulas and types kept separate.
- Minimal logic: $\to, \forall$ only. $x = y$ (Leibniz equality), $\exists$, $\vee$, $\wedge$ inductively defined (Martin-Löf).
- $\bot := (\mathrm{False} = \mathrm{True})$. Ex-falso-quodlibet: $\bot \to A$ provable.
- "Decorations" $\to^{\mathrm{nc}}$, $\forall^{\mathrm{nc}}$ (i) allow abstract theory (ii) remove unused data.

Relation of TCF to type theory

- Main difference: partial functionals are first class citizens.
- "Logic enriched": Formulas and types kept separate.
- Minimal logic: $\to, \forall$ only. $x = y$ (Leibniz equality), $\exists$, $\vee$, $\wedge$ inductively defined (Martin-Löf).
- $\bot := (\mathrm{False} = \mathrm{True})$. Ex-falso-quodlibet: $\bot \to A$ provable.
- "Decorations" $\to^{\mathrm{nc}}$, $\forall^{\mathrm{nc}}$ (i) allow abstract theory (ii) remove unused data.

Relation of TCF to type theory

- Main difference: partial functionals are first class citizens.
- "Logic enriched": Formulas and types kept separate.
- Minimal logic: $\to, \forall$ only. $x = y$ (Leibniz equality), $\exists$, $\vee$, $\wedge$ inductively defined (Martin-Löf).
- $\bot := (\mathrm{False} = \mathrm{True})$. Ex-falso-quodlibet: $\bot \to A$ provable.
- "Decorations" $\to^{\mathrm{nc}}$, $\forall^{\mathrm{nc}}$ (i) allow abstract theory (ii) remove unused data.

Case study: uniformly continuous functions (U. Berger)

- Formalization of an abstract theory of (uniformly) continuous real functions $f : I \to I \quad (I := [-1, 1])$.
- Let $\mathrm{C}f$ express that f is a continuous real function. Assume the abstract theory proves

$$\mathrm{C}f \to \forall_n \exists_m \underbrace{\forall_a \exists_b (f[I_{a,m}] \subseteq I_{b,n})}_{\mathrm{B}_{m,n} f} \quad \text{with } I_{b,n} := [b - \tfrac{1}{2^n}, b + \tfrac{1}{2^n}]$$

Then

$n \mapsto m$	modulus of (uniform) continuity (ω)
$n, a \mapsto b$	approximating rational function (h)

Case study: uniformly continuous functions (U. Berger)

- Formalization of an abstract theory of (uniformly) continuous real functions $f: I \to I \quad (I := [-1, 1])$.
- Let $\mathrm{C}f$ express that f is a continuous real function. Assume the abstract theory proves

$$\mathrm{C}f \to \forall_n \exists_m \underbrace{\forall_a \exists_b (f[I_{a,m}] \subseteq I_{b,n})}_{\mathrm{B}_{m,n} f} \quad \text{with } I_{b,n} := [b - \tfrac{1}{2^n}, b + \tfrac{1}{2^n}]$$

Then

$n \mapsto m$	modulus of (uniform) continuity (ω)
$n, a \mapsto b$	approximating rational function (h)

Case study: uniformly continuous functions (U. Berger)

- Formalization of an abstract theory of (uniformly) continuous real functions $f\colon I \to I \quad (I := [-1,1])$.
- Let $\mathrm{C}f$ express that f is a continuous real function. Assume the abstract theory proves

$$\mathrm{C}f \to \forall_n \exists_m \underbrace{\forall_a \exists_b (f[I_{a,m}] \subseteq I_{b,n})}_{\mathrm{B}_{m,n} f} \quad \text{with } I_{b,n} := [b - \tfrac{1}{2^n}, b + \tfrac{1}{2^n}]$$

Then

$n \mapsto m$	modulus of (uniform) continuity (ω)
$n, a \mapsto b$	approximating rational function (h)

Case study: uniformly continuous functions (U. Berger)

- Formalization of an abstract theory of (uniformly) continuous real functions $f\colon I \to I \quad (I := [-1,1])$.
- Let $\mathrm{C}f$ express that f is a continuous real function. Assume the abstract theory proves

$$\mathrm{C}f \to \forall_n \exists_m \underbrace{\forall_a \exists_b (f[I_{a,m}] \subseteq I_{b,n})}_{\mathrm{B}_{m,n} f} \quad \text{with } I_{b,n} := [b - \tfrac{1}{2^n}, b + \tfrac{1}{2^n}]$$

Then

$n \mapsto m$	modulus of (uniform) continuity (ω)
$n, a \mapsto b$	approximating rational function (h)

Read_X and its witnesses

Inductively define a predicate Read_X of arity (φ) by the clauses

$$\forall^{\mathrm{nc}}_f \forall_d (f[I] \subseteq I_d \to X(\mathrm{Out}_d \circ f) \to \mathrm{Read}_X f), \qquad (\mathrm{Read}_X)^+_0$$

$$\forall^{\mathrm{nc}}_f (\mathrm{Read}_X(f \circ \mathrm{In}_{-1}) \to \mathrm{Read}_X(f \circ \mathrm{In}_0) \to \mathrm{Read}_X(f \circ \mathrm{In}_1) \to \mathrm{Read}_X f). \qquad (\mathrm{Read}_X)^+_1$$

where $I_d = [\frac{d-1}{2}, \frac{d+1}{2}]$ $(d \in \{-1, 0, 1\})$ and

$$(\mathrm{Out}_d \circ f)(x) := 2f(x) - d, \qquad (f \circ \mathrm{In}_d)(x) := f(\frac{x+d}{2}).$$

Witnesses for $\mathrm{Read}_X f$: total ideals in

$$\mathbf{R}_\alpha := \mu_\xi(\mathsf{Put}^{\mathbf{SD} \to \alpha \to \xi}, \mathsf{Get}^{\xi \to \xi \to \xi \to \xi})$$

where $\mathbf{SD} := \{-1, 0, 1\}$.

Read_X and its witnesses

Inductively define a predicate Read_X of arity (φ) by the clauses

$$\forall^{\mathrm{nc}}_f \forall_d (f[I] \subseteq I_d \to X(\mathrm{Out}_d \circ f) \to \mathrm{Read}_X f), \qquad (\mathrm{Read}_X)^+_0$$

$$\forall^{\mathrm{nc}}_f (\mathrm{Read}_X(f \circ \mathrm{In}_{-1}) \to \mathrm{Read}_X(f \circ \mathrm{In}_0) \to \mathrm{Read}_X(f \circ \mathrm{In}_1) \to \mathrm{Read}_X f). \qquad (\mathrm{Read}_X)^+_1$$

where $I_d = [\frac{d-1}{2}, \frac{d+1}{2}]$ $(d \in \{-1, 0, 1\})$ and

$$(\mathrm{Out}_d \circ f)(x) := 2f(x) - d, \qquad (f \circ \mathrm{In}_d)(x) := f(\frac{x+d}{2}).$$

Witnesses for $\mathrm{Read}_X f$: total ideals in

$$\mathbf{R}_\alpha := \mu_\xi(\mathsf{Put}^{\mathbf{SD} \to \alpha \to \xi}, \mathsf{Get}^{\xi \to \xi \to \xi \to \xi})$$

where $\mathbf{SD} := \{-1, 0, 1\}$.

Read_X and its witnesses

Inductively define a predicate Read_X of arity (φ) by the clauses

$$\forall^{\mathrm{nc}}_f \forall_d (f[I] \subseteq I_d \to X(\mathrm{Out}_d \circ f) \to \mathrm{Read}_X f), \qquad (\mathrm{Read}_X)^+_0$$

$$\forall^{\mathrm{nc}}_f (\mathrm{Read}_X(f \circ \mathrm{In}_{-1}) \to \mathrm{Read}_X(f \circ \mathrm{In}_0) \to \mathrm{Read}_X(f \circ \mathrm{In}_1) \to \mathrm{Read}_X f). \qquad (\mathrm{Read}_X)^+_1$$

where $I_d = [\frac{d-1}{2}, \frac{d+1}{2}]$ $(d \in \{-1, 0, 1\})$ and

$$(\mathrm{Out}_d \circ f)(x) := 2f(x) - d, \qquad (f \circ \mathrm{In}_d)(x) := f(\frac{x+d}{2}).$$

Witnesses for $\mathrm{Read}_X f$: total ideals in

$$\mathbf{R}_\alpha := \mu_\xi(\mathsf{Put}^{\mathbf{SD} \to \alpha \to \xi}, \mathsf{Get}^{\xi \to \xi \to \xi \to \xi})$$

where $\mathbf{SD} := \{-1, 0, 1\}$.

Write, ${}^{\mathrm{co}}$Write and its witnesses

Nested inductive definition of a predicate Write of arity (φ):

$$\mathrm{Write}(\mathrm{Id}), \quad \forall^{\mathrm{nc}}_f(\mathrm{Read}_{\mathrm{Write}} f \to \mathrm{Write}\, f) \qquad (\mathrm{Id} \text{ identity function}).$$

Witnesses for $\mathrm{Write}\, f$: total ideals in

$$\mathbf{W} := \mu_\xi(\mathsf{Stop}^\xi, \mathsf{Cont}^{\mathbf{R}_\xi \to \xi}).$$

Define ${}^{\mathrm{co}}$Write, a companion predicate of Write, by

$$\forall^{\mathrm{nc}}_f({}^{\mathrm{co}}\mathrm{Write}\, f \to f = \mathrm{Id} \vee \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}} f). \qquad ({}^{\mathrm{co}}\mathrm{Write})^-$$

Witnesses for ${}^{\mathrm{co}}\mathrm{Write}\, f$: **W**-cototal $\mathbf{R}_{\mathbf{W}}$-total ideals t.

Write, $^{\mathrm{co}}$Write and its witnesses

Nested inductive definition of a predicate Write of arity (φ):

$\mathrm{Write}(\mathrm{Id}), \quad \forall^{\mathrm{nc}}_{f}(\mathrm{Read}_{\mathrm{Write}} f \to \mathrm{Write}\, f) \qquad (\mathrm{Id}$ identity function).

Witnesses for $\mathrm{Write}\, f$: total ideals in

$$\mathbf{W} := \mu_\xi(\mathsf{Stop}^\xi, \mathsf{Cont}^{\mathbf{R}_\xi \to \xi}).$$

Define $^{\mathrm{co}}\mathrm{Write}$, a companion predicate of Write, by

$$\forall^{\mathrm{nc}}_{f}({}^{\mathrm{co}}\mathrm{Write}\, f \to f = \mathrm{Id} \vee \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}} f). \qquad ({}^{\mathrm{co}}\mathrm{Write})^-$$

Witnesses for $^{\mathrm{co}}\mathrm{Write}\, f$: $\mathbf{W}$-cototal $\mathbf{R}_{\mathbf{W}}$-total ideals t.

Write, $^{\mathrm{co}}\mathrm{Write}$ and its witnesses

Nested inductive definition of a predicate Write of arity (φ):

$\mathrm{Write}(\mathrm{Id})$, $\quad \forall^{\mathrm{nc}}_f(\mathrm{Read}_{\mathrm{Write}} f \to \mathrm{Write}\, f)$ $\qquad$ (Id identity function).

Witnesses for $\mathrm{Write}\, f$: total ideals in

$$\mathbf{W} := \mu_\xi(\mathsf{Stop}^\xi, \mathsf{Cont}^{\mathbf{R}_\xi \to \xi}).$$

Define $^{\mathrm{co}}\mathrm{Write}$, a companion predicate of Write, by

$$\forall^{\mathrm{nc}}_f({}^{\mathrm{co}}\mathrm{Write}\, f \to f = \mathrm{Id} \lor \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}} f). \qquad ({}^{\mathrm{co}}\mathrm{Write})^-$$

Witnesses for $^{\mathrm{co}}\mathrm{Write}\, f$: $\mathbf{W}$-cototal $\mathbf{R}_{\mathbf{W}}$-total ideals t.

Write, $^{\text{co}}$Write and its witnesses

Nested inductive definition of a predicate Write of arity (φ):

$\text{Write}(\text{Id}), \quad \forall^{\text{nc}}_{f}(\text{Read}_{\text{Write}} f \to \text{Write}\, f) \qquad$ (Id identity function).

Witnesses for $\text{Write}\, f$: total ideals in

$$\mathbf{W} := \mu_\xi(\mathsf{Stop}^\xi, \mathsf{Cont}^{\mathbf{R}_\xi \to \xi}).$$

Define $^{\text{co}}$Write, a companion predicate of Write, by

$$\forall^{\text{nc}}_{f}({}^{\text{co}}\text{Write}\, f \to f = \text{Id} \vee \text{Read}_{{}^{\text{co}}\text{Write}} f). \qquad ({}^{\text{co}}\text{Write})^-$$

Witnesses for $^{\text{co}}\text{Write}\, f$: $\mathbf{W}$-cototal $\mathbf{R}_{\mathbf{W}}$-total ideals t.

Write, $^{\mathrm{co}}$Write and its witnesses

Nested inductive definition of a predicate Write of arity (φ):

$\mathrm{Write}(\mathrm{Id})$, $\quad \forall^{\mathrm{nc}}_{f}(\mathrm{Read}_{\mathrm{Write}} f \to \mathrm{Write}\, f)$ $\qquad$ (Id identity function).

Witnesses for $\mathrm{Write}\, f$: total ideals in

$$\mathbf{W} := \mu_\xi(\mathsf{Stop}^\xi, \mathsf{Cont}^{\mathbf{R}_\xi \to \xi}).$$

Define $^{\mathrm{co}}$Write, a companion predicate of Write, by

$$\forall^{\mathrm{nc}}_{f}({}^{\mathrm{co}}\mathrm{Write}\, f \to f = \mathrm{Id} \lor \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}} f). \qquad ({}^{\mathrm{co}}\mathrm{Write})^-$$

Witnesses for $^{\mathrm{co}}\mathrm{Write}\, f$: $\mathbf{W}$-cototal $\mathbf{R}_{\mathbf{W}}$-total ideals t.

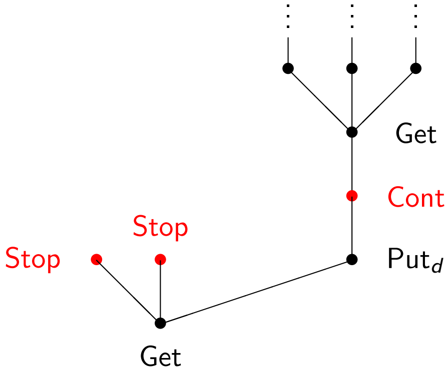
$\mathbf{W}$-cototal $\mathbf{R_W}$-total ideals

are possibly non well-founded trees t:

- Get-Put-part: well-founded,
- Stop-Cont-part: not necessarily well-founded.

$\mathbf{W}$-cototal $\mathbf{R_W}$-total ideals

are possibly non well-founded trees t:



- Get-Put-part: well-founded,
- Stop-Cont-part: not necessarily well-founded.

$\mathbf{W}$-cototal $\mathbf{R_W}$-total ideals as stream transformers

View them as read-write machines.

- Start at the root of the tree.
- At node $\mathrm{Put}_d t$, output the digit d, carry on with the tree t.
- At node Get $t_{-1}\ t_0\ t_1$, read a digit d from the input stream and continue with the tree t_d.
- At node Stop, return the rest of the input unprocessed as output.
- At node Cont t, continue with the tree t.

Output might be infinite, but $\mathbf{R_W}$-totality ensures that the machine can only read finitely many input digits before producing another output digit.

The machine represents a continuous function.

W-cototal $\mathbf{R_W}$-total ideals as stream transformers

View them as read-write machines.

- Start at the root of the tree.
- At node $\text{Put}_d t$, output the digit d, carry on with the tree t.
- At node Get t_{-1} t_0 t_1, read a digit d from the input stream and continue with the tree t_d.
- At node Stop, return the rest of the input unprocessed as output.
- At node Cont t, continue with the tree t.

Output might be infinite, but $\mathbf{R_W}$-totality ensures that the machine can only read finitely many input digits before producing another output digit.

The machine represents a continuous function.

W-cototal $\mathbf{R}_\mathbf{W}$-total ideals as stream transformers

View them as read-write machines.

- Start at the root of the tree.
- At node $\text{Put}_d t$, output the digit d, carry on with the tree t.
- At node Get t_{-1} t_0 t_1, read a digit d from the input stream and continue with the tree t_d.
- At node Stop, return the rest of the input unprocessed as output.
- At node Cont t, continue with the tree t.

Output might be infinite, but $\mathbf{R}_\mathbf{W}$-totality ensures that the machine can only read finitely many input digits before producing another output digit.

The machine represents a continuous function.

$\mathbf{W}$-cototal $\mathbf{R_W}$-total ideals as stream transformers

View them as read-write machines.

- Start at the root of the tree.
- At node $\mathsf{Put}_d\, t$, output the digit d, carry on with the tree t.
- At node Get t_{-1} t_0 t_1, read a digit d from the input stream and continue with the tree t_d.
- At node Stop, return the rest of the input unprocessed as output.
- At node Cont t, continue with the tree t.

Output might be infinite, but $\mathbf{R_W}$-totality ensures that the machine can only read finitely many input digits before producing another output digit.

The machine represents a continuous function.

W-cototal $\mathbf{R_W}$-total ideals as stream transformers

View them as read-write machines.

- Start at the root of the tree.
- At node $\text{Put}_d\, t$, output the digit d, carry on with the tree t.
- At node Get t_{-1} t_0 t_1, read a digit d from the input stream and continue with the tree t_d.
- At node Stop, return the rest of the input unprocessed as output.
- At node Cont t, continue with the tree t.

Output might be infinite, but $\mathbf{R_W}$-totality ensures that the machine can only read finitely many input digits before producing another output digit.

The machine represents a continuous function.

W-cototal $\mathbf{R_W}$-total ideals as stream transformers

View them as read-write machines.

- Start at the root of the tree.
- At node $\mathrm{Put}_d\, t$, output the digit d, carry on with the tree t.
- At node Get $t_{-1}\ t_0\ t_1$, read a digit d from the input stream and continue with the tree t_d.
- At node Stop, return the rest of the input unprocessed as output.
- At node Cont t, continue with the tree t.

Output might be infinite, but $\mathbf{R_W}$-totality ensures that the machine can only read finitely many input digits before producing another output digit.

The machine represents a continuous function.

W-cototal $\mathbf{R_W}$-total ideals as stream transformers

View them as read-write machines.

- Start at the root of the tree.
- At node $\mathsf{Put}_d\, t$, output the digit d, carry on with the tree t.
- At node Get $t_{-1}\ t_0\ t_1$, read a digit d from the input stream and continue with the tree t_d.
- At node Stop, return the rest of the input unprocessed as output.
- At node Cont t, continue with the tree t.

Output might be infinite, but $\mathbf{R_W}$-totality ensures that the machine can only read finitely many input digits before producing another output digit.

The machine represents a continuous function.

$\mathbf{W}$-cototal $\mathbf{R_W}$-total ideals as stream transformers

View them as read-write machines.

- Start at the root of the tree.
- At node $\mathrm{Put}_d\, t$, output the digit d, carry on with the tree t.
- At node Get t_{-1} t_0 t_1, read a digit d from the input stream and continue with the tree t_d.
- At node Stop, return the rest of the input unprocessed as output.
- At node Cont t, continue with the tree t.

Output might be infinite, but $\mathbf{R_W}$-totality ensures that the machine can only read finitely many input digits before producing another output digit.

The machine represents a continuous function.

$\mathbf{W}$-cototal $\mathbf{R_W}$-total ideals as stream transformers

View them as read-write machines.

- Start at the root of the tree.
- At node $\text{Put}_d\, t$, output the digit d, carry on with the tree t.
- At node Get t_{-1} t_0 t_1, read a digit d from the input stream and continue with the tree t_d.
- At node Stop, return the rest of the input unprocessed as output.
- At node Cont t, continue with the tree t.

Output might be infinite, but $\mathbf{R_W}$-totality ensures that the machine can only read finitely many input digits before producing another output digit.

The machine represents a continuous function.

$\mathrm{C}f$ implies ${}^{\mathrm{co}}\mathrm{Write}\, f$: informal proof

The greatest-fixed-point axiom $({}^{\mathrm{co}}\mathrm{Write})^+$ (coinduction) is

$$\forall^{\mathrm{nc}}_f (Q\, f \to \forall^{\mathrm{nc}}_f (Q\, f \to f = \mathrm{Id} \vee \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee Q} f) \to {}^{\mathrm{co}}\mathrm{Write}\, f).$$

Theorem [Type-1 u.c.f. into type-0 u.c.f.]. $\forall^{\mathrm{nc}}_f (\mathrm{C}f \to {}^{\mathrm{co}}\mathrm{Write}\, f)$.

Proof. Assume $\mathrm{C}f$. Use $({}^{\mathrm{co}}\mathrm{Write})^+$ with competitor C. Suffices $\forall^{\mathrm{nc}}_f (\mathrm{C}f \to f = \mathrm{Id} \vee \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee \mathrm{C}} f)$. Assume $\mathrm{C}f$, in particular $\mathrm{B}_{m,2} f := \forall_a \exists_b (f[I_{a,m}] \subseteq I_{b,2})$ for some m. Get rhs by Lemma 1.

Lemma 1. $\forall_m \forall^{\mathrm{nc}}_f (\mathrm{B}_{m,2} f \to \mathrm{C}f \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee \mathrm{C}} f)$.

Proof. Induction on m, using Lemma 2 in the base case.

Lemma 2 [FindSD]. $\forall^{\mathrm{nc}}_f (\mathrm{B}_{0,2} f \to \exists_d (f[I] \subseteq I_d))$.

Proof. Assume $\mathrm{B}_{0,2} f$. Then $f[I_{0,0}] \subseteq I_{b,2}$ for some b, by definition of $\mathrm{B}_{n,m}$. Have $b \le -\frac{1}{4}$, $-\frac{1}{4} \le b \le \frac{1}{4}$ or $\frac{1}{4} \le b$. Can determine either of $I_{b,2} \subseteq I_{-1}$, $I_{b,2} \subseteq I_0$ or $I_{b,2} \subseteq I_1$, hence $\exists_d (f[I] \subseteq I_d)$.

$\mathrm{C}f$ implies ${}^{\mathrm{co}}\mathrm{Write}\, f$: informal proof

The greatest-fixed-point axiom $({}^{\mathrm{co}}\mathrm{Write})^+$ (coinduction) is

$$\forall^{\mathrm{nc}}_f(Q\,f \to \forall^{\mathrm{nc}}_f(Q\,f \to f = \mathrm{Id} \vee \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee Q} f) \to {}^{\mathrm{co}}\mathrm{Write}\, f).$$

Theorem [Type-1 u.c.f. into type-0 u.c.f.]. $\forall^{\mathrm{nc}}_f(\mathrm{C}f \to {}^{\mathrm{co}}\mathrm{Write}\, f)$.

Proof. Assume $\mathrm{C}f$. Use $({}^{\mathrm{co}}\mathrm{Write})^+$ with competitor C. Suffices $\forall^{\mathrm{nc}}_f(\mathrm{C}f \to f = \mathrm{Id} \vee \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee \mathrm{C}} f)$. Assume $\mathrm{C}f$, in particular $\mathrm{B}_{m,2} f := \forall_a \exists_b (f[I_{a,m}] \subseteq I_{b,2})$ for some m. Get rhs by Lemma 1.

Lemma 1. $\forall_m \forall^{\mathrm{nc}}_f(\mathrm{B}_{m,2} f \to \mathrm{C}f \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee \mathrm{C}} f)$.

Proof. Induction on m, using Lemma 2 in the base case.

Lemma 2 [FindSD]. $\forall^{\mathrm{nc}}_f(\mathrm{B}_{0,2} f \to \exists_d (f[I] \subseteq I_d))$.

Proof. Assume $\mathrm{B}_{0,2} f$. Then $f[I_{0,0}] \subseteq I_{b,2}$ for some b, by definition of $\mathrm{B}_{n,m}$. Have $b \le -\frac{1}{4}$, $-\frac{1}{4} \le b \le \frac{1}{4}$ or $\frac{1}{4} \le b$. Can determine either of $I_{b,2} \subseteq I_{-1}$, $I_{b,2} \subseteq I_0$ or $I_{b,2} \subseteq I_1$, hence $\exists_d (f[I] \subseteq I_d)$.

$\mathrm{C}f$ implies ${}^{\mathrm{co}}\mathrm{Write}\, f$: informal proof

The greatest-fixed-point axiom $({}^{\mathrm{co}}\mathrm{Write})^+$ (coinduction) is

$$\forall^{\mathrm{nc}}_f(Q\,f \to \forall^{\mathrm{nc}}_f(Q\,f \to f = \mathrm{Id} \vee \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee Q} f) \to {}^{\mathrm{co}}\mathrm{Write}\, f).$$

Theorem [Type-1 u.c.f. into type-0 u.c.f.]. $\forall^{\mathrm{nc}}_f(\mathrm{C}f \to {}^{\mathrm{co}}\mathrm{Write}\, f)$.

Proof. Assume $\mathrm{C}f$. Use $({}^{\mathrm{co}}\mathrm{Write})^+$ with competitor C. Suffices $\forall^{\mathrm{nc}}_f(\mathrm{C}f \to f = \mathrm{Id} \vee \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee \mathrm{C}} f)$. Assume $\mathrm{C}f$, in particular $\mathrm{B}_{m,2}f := \forall_a \exists_b (f[I_{a,m}] \subseteq I_{b,2})$ for some m. Get rhs by Lemma 1.

Lemma 1. $\forall_m \forall^{\mathrm{nc}}_f(\mathrm{B}_{m,2}f \to \mathrm{C}f \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee \mathrm{C}} f)$.

Proof. Induction on m, using Lemma 2 in the base case.

Lemma 2 [FindSD]. $\forall^{\mathrm{nc}}_f(\mathrm{B}_{0,2}f \to \exists_d(f[I] \subseteq I_d))$.

Proof. Assume $\mathrm{B}_{0,2}f$. Then $f[I_{0,0}] \subseteq I_{b,2}$ for some b, by definition of $\mathrm{B}_{n,m}$. Have $b \le -\frac{1}{4}$, $-\frac{1}{4} \le b \le \frac{1}{4}$ or $\frac{1}{4} \le b$. Can determine either of $I_{b,2} \subseteq I_{-1}$, $I_{b,2} \subseteq I_0$ or $I_{b,2} \subseteq I_1$, hence $\exists_d(f[I] \subseteq I_d)$.

$\mathrm{C}f$ implies ${}^{\mathrm{co}}\mathrm{Write}\, f$: informal proof

The greatest-fixed-point axiom $({}^{\mathrm{co}}\mathrm{Write})^+$ (coinduction) is

$$\forall^{\mathrm{nc}}_f (Q\, f \to \forall^{\mathrm{nc}}_f (Q\, f \to f = \mathrm{Id} \lor \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \lor Q} f) \to {}^{\mathrm{co}}\mathrm{Write}\, f).$$

Theorem [Type-1 u.c.f. into type-0 u.c.f.]. $\forall^{\mathrm{nc}}_f (\mathrm{C}f \to {}^{\mathrm{co}}\mathrm{Write}\, f)$.

Proof. Assume $\mathrm{C}f$. Use $({}^{\mathrm{co}}\mathrm{Write})^+$ with competitor C. Suffices $\forall^{\mathrm{nc}}_f (\mathrm{C}f \to f = \mathrm{Id} \lor \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \lor \mathrm{C}} f)$. Assume $\mathrm{C}f$, in particular $\mathrm{B}_{m,2} f := \forall_a \exists_b (f[I_{a,m}] \subseteq I_{b,2})$ for some m. Get rhs by Lemma 1.

Lemma 1. $\forall_m \forall^{\mathrm{nc}}_f (\mathrm{B}_{m,2} f \to \mathrm{C}f \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \lor \mathrm{C}} f)$.

Proof. Induction on m, using Lemma 2 in the base case.

Lemma 2 [FindSD]. $\forall^{\mathrm{nc}}_f (\mathrm{B}_{0,2} f \to \exists_d (f[I] \subseteq I_d))$.

Proof. Assume $\mathrm{B}_{0,2} f$. Then $f[I_{0,0}] \subseteq I_{b,2}$ for some b, by definition of $\mathrm{B}_{n,m}$. Have $b \le -\frac{1}{4}$, $-\frac{1}{4} \le b \le \frac{1}{4}$ or $\frac{1}{4} \le b$. Can determine either of $I_{b,2} \subseteq I_{-1}$, $I_{b,2} \subseteq I_0$ or $I_{b,2} \subseteq I_1$, hence $\exists_d (f[I] \subseteq I_d)$.

$\mathrm{C}f$ implies ${}^{\mathrm{co}}\mathrm{Write}\, f$: informal proof

The greatest-fixed-point axiom $({}^{\mathrm{co}}\mathrm{Write})^+$ (coinduction) is

$$\forall^{\mathrm{nc}}_f (Q\, f \to \forall^{\mathrm{nc}}_f (Q\, f \to f = \mathrm{Id} \vee \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee Q} f) \to {}^{\mathrm{co}}\mathrm{Write}\, f).$$

Theorem [Type-1 u.c.f. into type-0 u.c.f.]. $\forall^{\mathrm{nc}}_f (\mathrm{C}f \to {}^{\mathrm{co}}\mathrm{Write}\, f)$.

Proof. Assume $\mathrm{C}f$. Use $({}^{\mathrm{co}}\mathrm{Write})^+$ with competitor C. Suffices $\forall^{\mathrm{nc}}_f (\mathrm{C}f \to f = \mathrm{Id} \vee \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee \mathrm{C}} f)$. Assume $\mathrm{C}f$, in particular $\mathrm{B}_{m,2} f := \forall_a \exists_b (f[I_{a,m}] \subseteq I_{b,2})$ for some m. Get rhs by Lemma 1.

Lemma 1. $\forall_m \forall^{\mathrm{nc}}_f (\mathrm{B}_{m,2} f \to \mathrm{C}f \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee \mathrm{C}} f)$.

Proof. Induction on m, using Lemma 2 in the base case.

Lemma 2 [FindSD]. $\forall^{\mathrm{nc}}_f (\mathrm{B}_{0,2} f \to \exists_d (f[I] \subseteq I_d))$.

Proof. Assume $\mathrm{B}_{0,2} f$. Then $f[I_{0,0}] \subseteq I_{b,2}$ for some b, by definition of $\mathrm{B}_{n,m}$. Have $b \le -\frac{1}{4}$, $-\frac{1}{4} \le b \le \frac{1}{4}$ or $\frac{1}{4} \le b$. Can determine either of $I_{b,2} \subseteq I_{-1}$, $I_{b,2} \subseteq I_0$ or $I_{b,2} \subseteq I_1$, hence $\exists_d (f[I] \subseteq I_d)$.

$\mathrm{C}f$ implies ${}^{\mathrm{co}}\mathrm{Write}\, f$: informal proof

The greatest-fixed-point axiom $({}^{\mathrm{co}}\mathrm{Write})^{+}$ (coinduction) is

$$\forall^{\mathrm{nc}}_{f}(Q\,f \to \forall^{\mathrm{nc}}_{f}(Q\,f \to f = \mathrm{Id} \vee \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee Q} f) \to {}^{\mathrm{co}}\mathrm{Write}\, f).$$

Theorem [Type-1 u.c.f. into type-0 u.c.f.]. $\forall^{\mathrm{nc}}_{f}(\mathrm{C}f \to {}^{\mathrm{co}}\mathrm{Write}\, f)$.

Proof. Assume $\mathrm{C}f$. Use $({}^{\mathrm{co}}\mathrm{Write})^{+}$ with competitor C. Suffices $\forall^{\mathrm{nc}}_{f}(\mathrm{C}f \to f = \mathrm{Id} \vee \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee \mathrm{C}} f)$. Assume $\mathrm{C}f$, in particular $\mathrm{B}_{m,2} f := \forall_{a} \exists_{b}(f[I_{a,m}] \subseteq I_{b,2})$ for some m. Get rhs by Lemma 1.

Lemma 1. $\forall_{m} \forall^{\mathrm{nc}}_{f}(\mathrm{B}_{m,2} f \to \mathrm{C}f \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee \mathrm{C}} f)$.

Proof. Induction on m, using Lemma 2 in the base case.

Lemma 2 [FindSD]. $\forall^{\mathrm{nc}}_{f}(\mathrm{B}_{0,2} f \to \exists_{d}(f[I] \subseteq I_{d}))$.

Proof. Assume $\mathrm{B}_{0,2} f$. Then $f[I_{0,0}] \subseteq I_{b,2}$ for some b, by definition of $\mathrm{B}_{n,m}$. Have $b \le -\frac{1}{4}$, $-\frac{1}{4} \le b \le \frac{1}{4}$ or $\frac{1}{4} \le b$. Can determine either of $I_{b,2} \subseteq I_{-1}$, $I_{b,2} \subseteq I_{0}$ or $I_{b,2} \subseteq I_{1}$, hence $\exists_{d}(f[I] \subseteq I_{d})$.

```
[oh](CoRec (nat=>nat@@(rat=>rat))=>algwrite)oh
 ([oh0]Inr((Rec nat=>..[type]..)
       left(oh0(Succ(Succ Zero)))
       ([g,oh1] [let sd (cFindSd(g 0))
           (Put sd
           (InR([n]left(oh1(Succ n))@
                ([a]2*right(oh1(Succ n))a-SDToInt sd))))])
       ([n,st,g,oh1]
         Get
         (st([a]g((a+IntN 1)/2))
          ([n0]left(oh1 n0)@
           ([a]right(oh1 n0)((a+IntN 1)/2))))
         (st([a]g(a/2))([n0]left(oh1 n0)@
                       ([a]right(oh1 n0)(a/2))))
         (st([a]g((a+1)/2))([n0]left(oh1 n0)@
                           ([a]right(oh1 n0)((a+1)/2)))))
       right(oh0(Succ(Succ Zero)))
       oh0))
```

Corecursion

The corecursion operator ${}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{W}}$ has type

$$\tau \to (\tau \to \mathbf{U} + \mathbf{R}_{\mathbf{W}+\tau}) \to \mathbf{W}.$$

Conversion rule

$$
\begin{aligned}
{}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{W}} NM \mapsto [&\textbf{case } (MN)^{\mathbf{U}+\mathbf{R}(\mathbf{W}+\tau)} \textbf{ of} \\
&\mathrm{Inl}\ \_ \mapsto \mathsf{Stop} \mid \\
&\mathrm{Inr}\ x \mapsto \mathsf{Cont}(\mathcal{M}^{\mathbf{W}}_{\mathbf{R}(\mathbf{W}+\tau)}(\lambda_p[\textbf{case } p^{\mathbf{W}+\tau} \textbf{ of} \\
&\qquad\qquad\qquad\qquad \mathrm{Inl}\ y^{\mathbf{W}} \mapsto y \mid \\
&\qquad\qquad\qquad\qquad \mathrm{Inr}\ z^{\tau} \mapsto {}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{W}} zM]) \\
&\qquad\qquad\qquad x^{\mathbf{R}(\mathbf{W}+\tau)}]
\end{aligned}
$$

with $\mathcal{M}$ a “map”-operator.

- Here τ is $\mathbf{N} \to \mathbf{N} \times (\mathbf{Q} \to \mathbf{Q})$, for pairs of $\omega \colon \mathbf{N} \to \mathbf{N}$ and $h \colon \mathbf{N} \to \mathbf{Q} \to \mathbf{Q}$ (variable name `oh`).
- No termination; translate into Haskell for evaluation.

Corecursion

The corecursion operator ${}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{W}}$ has type

$$\tau \to (\tau \to \mathbf{U} + \mathbf{R}_{\mathbf{W}+\tau}) \to \mathbf{W}.$$

Conversion rule

$$\begin{aligned}
{}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{W}} NM \mapsto [&\mathbf{case}\ (MN)^{\mathbf{U}+\mathbf{R}(\mathbf{W}+\tau)}\ \mathbf{of} \\
&\mathrm{Inl}\ _ \mapsto \mathsf{Stop} \mid \\
&\mathrm{Inr}\ x \mapsto \mathsf{Cont}(\mathcal{M}^{\mathbf{W}}_{\mathbf{R}(\mathbf{W}+\tau)}(\lambda_p[\mathbf{case}\ p^{\mathbf{W}+\tau}\ \mathbf{of} \\
&\qquad \mathrm{Inl}\ y^{\mathbf{W}} \mapsto y \mid \\
&\qquad \mathrm{Inr}\ z^{\tau} \mapsto {}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{W}} zM]) \\
&\quad x^{\mathbf{R}(\mathbf{W}+\tau)}]
\end{aligned}$$

with $\mathcal{M}$ a "map"-operator.

- Here τ is $\mathbf{N} \to \mathbf{N} \times (\mathbf{Q} \to \mathbf{Q})$, for pairs of $\omega \colon \mathbf{N} \to \mathbf{N}$ and $h \colon \mathbf{N} \to \mathbf{Q} \to \mathbf{Q}$ (variable name oh).
- No termination; translate into Haskell for evaluation.

Corecursion

The corecursion operator ${}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{W}}$ has type

$$\tau \to (\tau \to \mathbf{U} + \mathbf{R}_{\mathbf{W}+\tau}) \to \mathbf{W}.$$

Conversion rule

$$\begin{aligned}
{}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{W}} NM \mapsto [&\textbf{case } (MN)^{\mathbf{U}+\mathbf{R}(\mathbf{W}+\tau)} \textbf{ of} \\
&\mathrm{Inl}\ _ \mapsto \mathsf{Stop} \mid \\
&\mathrm{Inr}\ x \mapsto \mathsf{Cont}(\mathcal{M}^{\mathbf{W}}_{\mathbf{R}(\mathbf{W}+\tau)}(\lambda_p[\textbf{case } p^{\mathbf{W}+\tau} \textbf{ of} \\
&\qquad \mathrm{Inl}\ y^{\mathbf{W}} \mapsto y \mid \\
&\qquad \mathrm{Inr}\ z^{\tau} \mapsto {}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{W}} zM]) \\
&\qquad x^{\mathbf{R}(\mathbf{W}+\tau)}]
\end{aligned}$$

with $\mathcal{M}$ a "map"-operator.

- Here τ is $\mathbf{N} \to \mathbf{N} \times (\mathbf{Q} \to \mathbf{Q})$, for pairs of $\omega\colon \mathbf{N} \to \mathbf{N}$ and $h\colon \mathbf{N} \to \mathbf{Q} \to \mathbf{Q}$ (variable name oh).
- No termination; translate into Haskell for evaluation.

Corecursion

The corecursion operator ${}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{W}}$ has type

$$\tau \to (\tau \to \mathbf{U} + \mathbf{R}_{\mathbf{W}+\tau}) \to \mathbf{W}.$$

Conversion rule

$$\begin{aligned}
{}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{W}} NM \mapsto [\textbf{case}\ (MN)^{\mathbf{U}+\mathbf{R}(\mathbf{W}+\tau)}\ \textbf{of}\ & \\
\mathrm{Inl}\ _ \mapsto \mathsf{Stop} \mid & \\
\mathrm{Inr}\ x \mapsto \mathsf{Cont}(\mathcal{M}^{\mathbf{W}}_{\mathbf{R}(\mathbf{W}+\tau)}(\lambda_p[\textbf{case}\ p^{\mathbf{W}+\tau}\ \textbf{of}\ & \\
\mathrm{Inl}\ y^{\mathbf{W}} \mapsto y \mid & \\
\mathrm{Inr}\ z^{\tau} \mapsto {}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{W}} zM]) & \\
x^{\mathbf{R}(\mathbf{W}+\tau)}] &
\end{aligned}$$

with $\mathcal{M}$ a "map"-operator.

- Here τ is $\mathbf{N} \to \mathbf{N} \times (\mathbf{Q} \to \mathbf{Q})$, for pairs of $\omega \colon \mathbf{N} \to \mathbf{N}$ and $h \colon \mathbf{N} \to \mathbf{Q} \to \mathbf{Q}$ (variable name `oh`).
- No termination; translate into Haskell for evaluation.

Corecursion

The corecursion operator ${}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{W}}$ has type

$$\tau \to (\tau \to \mathbf{U} + \mathbf{R}_{\mathbf{W}+\tau}) \to \mathbf{W}.$$

Conversion rule

$$\begin{aligned}
{}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{W}} NM \mapsto {} & [\textbf{case } (MN)^{\mathbf{U}+\mathbf{R}(\mathbf{W}+\tau)} \textbf{ of} \\
& \mathrm{Inl}\ _ \mapsto \mathsf{Stop} \mid \\
& \mathrm{Inr}\ x \mapsto \mathsf{Cont}(\mathcal{M}^{\mathbf{W}}_{\mathbf{R}(\mathbf{W}+\tau)}(\lambda_p[\textbf{case } p^{\mathbf{W}+\tau} \textbf{ of} \\
& \qquad \mathrm{Inl}\ y^{\mathbf{W}} \mapsto y \mid \\
& \qquad \mathrm{Inr}\ z^{\tau} \mapsto {}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{W}} zM]) \\
& \qquad x^{\mathbf{R}(\mathbf{W}+\tau)}]
\end{aligned}$$

with $\mathcal{M}$ a “map”-operator.

- Here τ is $\mathbf{N} \to \mathbf{N} \times (\mathbf{Q} \to \mathbf{Q})$, for pairs of $\omega\colon \mathbf{N} \to \mathbf{N}$ and $h\colon \mathbf{N} \to \mathbf{Q} \to \mathbf{Q}$ (variable name `oh`).
- No termination; translate into Haskell for evaluation.

Conclusion

TCF (theory of computable functionals) as a possible foundation for exact real arithmetic.

- Simply typed theory, with “lazy” free algebras as base types ($\Rightarrow$ constructors are injective and have disjoint ranges).
- Variables range over partial continuous functionals.
- Constants denote computable functionals (:= r.e. ideals).
- Minimal logic ($\to$, $\forall$), plus inductive & coinductive definitions.
- Computational content in abstract theories.
- Decorations ($\to, \forall$ and $\to^{\mathrm{nc}}, \forall^{\mathrm{nc}}$) for fine-tuning.

References

- U. Berger, From coinductive proofs to exact real arithmetic. CSL 2009.
- K. Miyamoto and H.S., Program extraction in exact real arithmetic. To appear, MSCS.
- K. Miyamoto, F. Nordvall Forsberg and H.S., Program extraction from nested definitions. ITP 2013.
- H.S. and S.S. Wainer, Proofs and Computations. Perspectives in Logic, ASL & Cambridge UP, 2012.