A theory of computable functionals

Helmut Schwichtenberg

Mathematisches Institut, LMU, München

Würzburg, 26. November 2024

Computational content of proofs

- Proofs may have computational content.
- One can extract it and obtains a term ($\sim$ program).
- The correctness of this term ($\sim$ program) can be proved.

This correctness proof is a formal one and within the underlying theory. It can be automatically generated.

What is a proof? Need (i) a language and (ii) logic.

Computational content of proofs

- Proofs may have computational content.
- One can extract it and obtains a term ($\sim$ program).
- The correctness of this term ($\sim$ program) can be proved.

This correctness proof is a formal one and within the underlying theory. It can be automatically generated.

What is a proof? Need (i) a language and (ii) logic.

Computational content of proofs

- Proofs may have computational content.
- One can extract it and obtains a term ($\sim$ program).
- The correctness of this term ($\sim$ program) can be proved.

This correctness proof is a formal one and within the underlying theory. It can be automatically generated.

What is a proof? Need (i) a language and (ii) logic.

Computational content of proofs

- Proofs may have computational content.
- One can extract it and obtains a term ($\sim$ program).
- The correctness of this term ($\sim$ program) can be proved.

This correctness proof is a formal one and within the underlying theory. It can be automatically generated.

What is a proof? Need (i) a language and (ii) logic.

Computational content of proofs

- Proofs may have computational content.
- One can extract it and obtains a term ($\sim$ program).
- The correctness of this term ($\sim$ program) can be proved.

This correctness proof is a formal one and within the underlying theory. It can be automatically generated.

What is a proof? Need (i) a language and (ii) logic.

Language

- Functions of (simple) types, defined by equations.
- Predicates, which are inductively / coinductively defined, by clauses and least / greatest fixed point axioms.

Predicates are marked as

c.r.	computationally relevant, or
n.c.	non-computational.

Language

- Functions of (simple) types, defined by equations.
- Predicates, which are inductively / coinductively defined, by clauses and least / greatest fixed point axioms.

Predicates are marked as

c.r.	computationally relevant, or
n.c.	non-computational.

Language

- Functions of (simple) types, defined by equations.
- Predicates, which are inductively / coinductively defined, by clauses and least / greatest fixed point axioms.

Predicates are marked as

c.r.	computationally relevant, or
n.c.	non-computational.

Logic

- A constructive extension of classical logic, by adding "strong" variants of $\vee, \exists$ to the classical $\tilde{\vee}, \tilde{\exists}$:

$$A \mathbin{\tilde{\vee}} B := (\neg A \to \neg B \to \bot), \qquad \tilde{\exists}_x A := \neg \forall_x \neg A.$$

- In proof trees (natural deduction) call subtrees with an n.c. end formula "nc-parts". Ignore c.r. and n.c. decorations there.

Logic

- A constructive extension of classical logic, by adding "strong" variants of $\vee, \exists$ to the classical $\tilde{\vee}, \tilde{\exists}$:

$$A \mathbin{\tilde{\vee}} B := (\neg A \to \neg B \to \bot), \qquad \tilde{\exists}_x A := \neg \forall_x \neg A.$$

- In proof trees (natural deduction) call subtrees with an n.c. end formula "nc-parts". Ignore c.r. and n.c. decorations there.

- What is a proof? We need a theory.
- Since we are interested in the computational content of proofs, it seems best to look for a theory describing a concrete model:
- Scott-Ershov model of partial continuous functionals[1]. Idea: Infinite objects ("ideals") given by their finite approximations.
- Ideals: "consistent" and "deductively closed" sets of "tokens".
- Tokens at base types: "constructor trees" with possibly $*$.

[1] Dana Scott, Outline of a mathematical theory of computation, 1970, and Yuri Ershov, Model C of partial continuous functionals, 1984

- What is a proof? We need a theory.
- Since we are interested in the computational content of proofs, it seems best to look for a theory describing a concrete model:
- Scott-Ershov model of partial continuous functionals[1]. Idea: Infinite objects ("ideals") given by their finite approximations.
- Ideals: "consistent" and "deductively closed" sets of "tokens".
- Tokens at base types: "constructor trees" with possibly $*$.

[1] Dana Scott, Outline of a mathematical theory of computation, 1970, and Yuri Ershov, Model C of partial continuous functionals, 1984

- What is a proof? We need a theory.
- Since we are interested in the computational content of proofs, it seems best to look for a theory describing a concrete model:
- Scott-Ershov model of partial continuous functionals[1]. Idea: Infinite objects ("ideals") given by their finite approximations.
- Ideals: "consistent" and "deductively closed" sets of "tokens".
- Tokens at base types: "constructor trees" with possibly $*$.

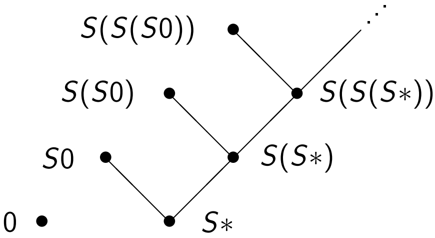
[1] Dana Scott, Outline of a mathematical theory of computation, 1970, and Yuri Ershov, Model C of partial continuous functionals, 1984

- What is a proof? We need a theory.
- Since we are interested in the computational content of proofs, it seems best to look for a theory describing a concrete model:
- Scott-Ershov model of partial continuous functionals[1]. Idea: Infinite objects ("ideals") given by their finite approximations.
- Ideals: "consistent" and "deductively closed" sets of "tokens".
- Tokens at base types: "constructor trees" with possibly $*$.

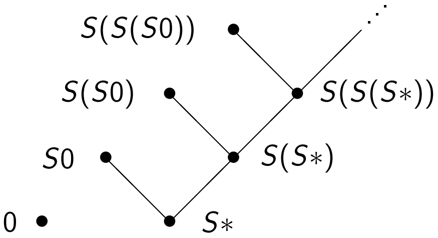
[1] Dana Scott, Outline of a mathematical theory of computation, 1970, and Yuri Ershov, Model C of partial continuous functionals, 1984

- What is a proof? We need a theory.
- Since we are interested in the computational content of proofs, it seems best to look for a theory describing a concrete model:
- Scott-Ershov model of partial continuous functionals[1]. Idea: Infinite objects ("ideals") given by their finite approximations.
- Ideals: "consistent" and "deductively closed" sets of "tokens".
- Tokens at base types: "constructor trees" with possibly $*$.

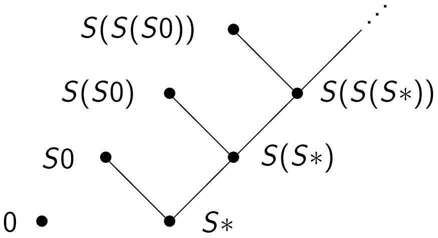
[1] Dana Scott, Outline of a mathematical theory of computation, 1970, and Yuri Ershov, Model *C* of partial continuous functionals, 1984



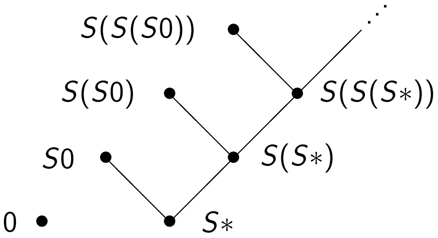
- $\{S0, S(S*)\}$ is inconsistent.
- $\{S*, S(S*)\}$ is an ideal.
- $\{S*, S(S*), S(S0)\}$ is an ideal ("total").
- $\{S*, S(S*), S(S(S*)), \dots\}$ is an infinite ideal ("cototal").



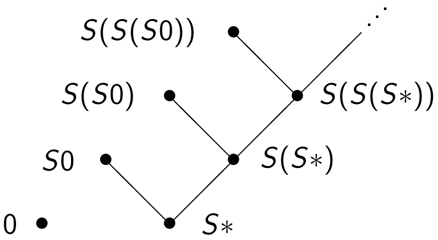
- $\{S0, S(S*)\}$ is inconsistent.
- $\{S*, S(S*)\}$ is an ideal.
- $\{S*, S(S*), S(S0)\}$ is an ideal ("total").
- $\{S*, S(S*), S(S(S*)), \dots\}$ is an infinite ideal ("cototal").



- $\{S0, S(S*)\}$ is inconsistent.
- $\{S*, S(S*)\}$ is an ideal.
- $\{S*, S(S*), S(S0)\}$ is an ideal ("total").
- $\{S*, S(S*), S(S(S*)), \dots\}$ is an infinite ideal ("cototal").



- $\{S0, S(S*)\}$ is inconsistent.
- $\{S*, S(S*)\}$ is an ideal.
- $\{S*, S(S*), S(S0)\}$ is an ideal ("total").
- $\{S*, S(S*), S(S(S*)), \dots\}$ is an infinite ideal ("cototal").



- $\{S0, S(S*)\}$ is inconsistent.
- $\{S*, S(S*)\}$ is an ideal.
- $\{S*, S(S*), S(S0)\}$ is an ideal ("total").
- $\{S*, S(S*), S(S(S*)), \dots\}$ is an infinite ideal ("cototal").

Ideals at function types

- can be partial,
- are continuous: for every "formal neighborhood" V of $f(x)$ we can find a formal neighborhood U of x with $f[U] \subseteq V$, and
- are computable iff they are given by a recursively enumerable set of tokens.

Ideals at function types

- can be partial,
- are continuous: for every "formal neighborhood" V of $f(x)$ we can find a formal neighborhood U of x with $f[U] \subseteq V$, and
- are computable iff they are given by a recursively enumerable set of tokens.

Ideals at function types

- can be partial,
- are continuous: for every "formal neighborhood" V of $f(x)$ we can find a formal neighborhood U of x with $f[U] \subseteq V$, and
- are computable iff they are given by a recursively enumerable set of tokens.

A common extension T^+ of Gödel's T and Plotkin's PCF

Terms: built from (typed) variables and constants (constructors C or defined constants D) by abstraction and application:

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

Examples: Decidable equality $=_{\mathbb{N}} : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$

$$\begin{aligned} (0 =_{\mathbb{N}} 0) &= \mathtt{tt}, & (Sn =_{\mathbb{N}} 0) &= \mathtt{ff}, \\ (0 =_{\mathbb{N}} Sm) &= \mathtt{ff}, & (Sn =_{\mathbb{N}} Sm) &= (n =_{\mathbb{N}} m). \end{aligned}$$

Recursion $\mathcal{R}^\tau_{\mathbb{N}} : \mathbb{N} \to \tau \to (\mathbb{N} \to \tau \to \tau) \to \tau$.

$$\begin{aligned} \mathcal{R}^\tau_{\mathbb{N}} 0af &= a, \\ \mathcal{R}^\tau_{\mathbb{N}} (Sn)af &= fn(\mathcal{R}^\tau_{\mathbb{N}} naf). \end{aligned}$$

A common extension T^+ of Gödel's T and Plotkin's PCF

Terms: built from (typed) variables and constants (constructors C or defined constants D) by abstraction and application:

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

Examples: Decidable equality $=_\mathbb{N} \colon \mathbb{N} \to \mathbb{N} \to \mathbb{B}$

$$(0 =_\mathbb{N} 0) = \mathtt{tt}, \qquad (Sn =_\mathbb{N} 0) = \mathtt{ff},$$
$$(0 =_\mathbb{N} Sm) = \mathtt{ff}, \qquad (Sn =_\mathbb{N} Sm) = (n =_\mathbb{N} m).$$

Recursion $\mathcal{R}^\tau_\mathbb{N} \colon \mathbb{N} \to \tau \to (\mathbb{N} \to \tau \to \tau) \to \tau$.

$$\mathcal{R}^\tau_\mathbb{N} 0 a f = a,$$
$$\mathcal{R}^\tau_\mathbb{N} (Sn) a f = f n (\mathcal{R}^\tau_\mathbb{N} n a f).$$

A common extension T^+ of Gödel's T and Plotkin's PCF

Terms: built from (typed) variables and constants (constructors C or defined constants D) by abstraction and application:

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

Examples: Decidable equality $=_\mathbb{N} \colon \mathbb{N} \to \mathbb{N} \to \mathbb{B}$

$$(0 =_\mathbb{N} 0) = \mathrm{tt}, \qquad (Sn =_\mathbb{N} 0) = \mathrm{ff},$$
$$(0 =_\mathbb{N} Sm) = \mathrm{ff}, \qquad (Sn =_\mathbb{N} Sm) = (n =_\mathbb{N} m).$$

Recursion $\mathcal{R}^\tau_\mathbb{N} \colon \mathbb{N} \to \tau \to (\mathbb{N} \to \tau \to \tau) \to \tau$.

$$\mathcal{R}^\tau_\mathbb{N} 0 a f = a,$$
$$\mathcal{R}^\tau_\mathbb{N} (Sn) a f = f n (\mathcal{R}^\tau_\mathbb{N} n a f).$$

A common extension T^+ of Gödel's T and Plotkin's PCF

Terms: built from (typed) variables and constants (constructors C or defined constants D) by abstraction and application:

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

Examples: Decidable equality $=_\mathbb{N} \colon \mathbb{N} \to \mathbb{N} \to \mathbb{B}$

$$\begin{aligned} (0 =_\mathbb{N} 0) &= \mathrm{tt}, & (Sn =_\mathbb{N} 0) &= \mathrm{ff}, \\ (0 =_\mathbb{N} Sm) &= \mathrm{ff}, & (Sn =_\mathbb{N} Sm) &= (n =_\mathbb{N} m). \end{aligned}$$

Recursion $\mathcal{R}^\tau_\mathbb{N} \colon \mathbb{N} \to \tau \to (\mathbb{N} \to \tau \to \tau) \to \tau$.

$$\begin{aligned} \mathcal{R}^\tau_\mathbb{N} 0 a f &= a, \\ \mathcal{R}^\tau_\mathbb{N} (Sn) a f &= f n (\mathcal{R}^\tau_\mathbb{N} n a f). \end{aligned}$$

Predicates and formulas

$$P, Q ::= X \mid \{\vec{x} \mid A\} \mid I(\vec{\rho}, \vec{P}) \mid {}^{\mathrm{co}}I(\vec{\rho}, \vec{P}) \qquad \text{(predicates)},$$
$$A, B ::= P\vec{t} \mid A \to B \mid \forall_x A \qquad \text{(formulas)}.$$

The missing logical connectives $\land, \lor, \exists$ are inductively defined.
Totality $T_{\mathbb{N}}$ is inductively defined as the least fixed point (lfp) of the clauses

$$0 \in T_{\mathbb{N}}, \qquad n \in T_{\mathbb{N}} \to Sn \in T_{\mathbb{N}}.$$

Cototality ${}^{\mathrm{co}}T_{\mathbb{N}}$ is coinductively defined as the greatest fixed point (gfp) of its closure axiom

$$n \in {}^{\mathrm{co}}T_{\mathbb{N}} \to n \equiv 0 \lor \exists_{n'}(n' \in {}^{\mathrm{co}}T_{\mathbb{N}} \land n \equiv Sn').$$

Predicates and formulas

$$P, Q ::= X \mid \{ \vec{x} \mid A \} \mid I(\vec{\rho}, \vec{P}\,) \mid {}^{\mathrm{co}}I(\vec{\rho}, \vec{P}\,) \qquad \text{(predicates)},$$
$$A, B ::= P\vec{t} \mid A \to B \mid \forall_x A \qquad \text{(formulas)}.$$

The missing logical connectives $\wedge, \vee, \exists$ are inductively defined.
Totality $T_{\mathbb{N}}$ is inductively defined as the least fixed point (lfp) of the clauses

$$0 \in T_{\mathbb{N}}, \qquad n \in T_{\mathbb{N}} \to Sn \in T_{\mathbb{N}}.$$

Cototality ${}^{\mathrm{co}}T_{\mathbb{N}}$ is coinductively defined as the greatest fixed point (gfp) of its closure axiom

$$n \in {}^{\mathrm{co}}T_{\mathbb{N}} \to n \equiv 0 \vee \exists_{n'}(n' \in {}^{\mathrm{co}}T_{\mathbb{N}} \wedge n \equiv Sn').$$

Predicates and formulas

$$P, Q ::= X \mid \{\, \vec{x} \mid A \,\} \mid I(\vec{\rho}, \vec{P}\,) \mid {}^{\mathrm{co}}I(\vec{\rho}, \vec{P}\,) \qquad \text{(predicates)},$$
$$A, B ::= P\vec{t} \mid A \to B \mid \forall_x A \qquad \text{(formulas)}.$$

The missing logical connectives $\land, \lor, \exists$ are inductively defined.

Totality $T_{\mathbb{N}}$ is inductively defined as the least fixed point (lfp) of the clauses

$$0 \in T_{\mathbb{N}}, \qquad n \in T_{\mathbb{N}} \to Sn \in T_{\mathbb{N}}.$$

Cototality ${}^{\mathrm{co}}T_{\mathbb{N}}$ is coinductively defined as the greatest fixed point (gfp) of its closure axiom

$$n \in {}^{\mathrm{co}}T_{\mathbb{N}} \to n \equiv 0 \lor \exists_{n'}(n' \in {}^{\mathrm{co}}T_{\mathbb{N}} \land n \equiv Sn').$$

Predicates and formulas

$$P, Q ::= X \mid \{\,\vec{x} \mid A\,\} \mid I(\vec{\rho}, \vec{P}\,) \mid {}^{\mathrm{co}}I(\vec{\rho}, \vec{P}\,) \qquad \text{(predicates)},$$
$$A, B ::= P\vec{t} \mid A \to B \mid \forall_x A \qquad \text{(formulas)}.$$

The missing logical connectives $\wedge, \vee, \exists$ are inductively defined.
Totality $T_{\mathbb{N}}$ is inductively defined as the least fixed point (lfp) of the clauses

$$0 \in T_{\mathbb{N}}, \qquad n \in T_{\mathbb{N}} \to Sn \in T_{\mathbb{N}}.$$

Cototality ${}^{\mathrm{co}}T_{\mathbb{N}}$ is coinductively defined as the greatest fixed point (gfp) of its closure axiom

$$n \in {}^{\mathrm{co}}T_{\mathbb{N}} \to n \equiv 0 \vee \exists_{n'}(n' \in {}^{\mathrm{co}}T_{\mathbb{N}} \wedge n \equiv Sn').$$

Predicates and formulas

$$
\begin{aligned}
P, Q &::= X \mid \{\, \vec{x} \mid A \,\} \mid I(\vec{\rho}, \vec{P}\,) \mid {}^{\mathrm{co}}I(\vec{\rho}, \vec{P}\,) && \text{(predicates)},\\
A, B &::= P\vec{t} \mid A \to B \mid \forall_x A && \text{(formulas)}.
\end{aligned}
$$

The missing logical connectives $\land, \lor, \exists$ are inductively defined.
Totality $T_{\mathbb{N}}$ is inductively defined as the least fixed point (lfp) of the clauses

$$0 \in T_{\mathbb{N}}, \qquad n \in T_{\mathbb{N}} \to Sn \in T_{\mathbb{N}}.$$

Cototality ${}^{\mathrm{co}}T_{\mathbb{N}}$ is coinductively defined as the greatest fixed point (gfp) of its closure axiom

$$n \in {}^{\mathrm{co}}T_{\mathbb{N}} \to n \equiv 0 \lor \exists_{n'}(n' \in {}^{\mathrm{co}}T_{\mathbb{N}} \land n \equiv Sn').$$

Partiality

- Defined functionals D (and hence terms) can be partial.
- Many D's are total (map total arguments into total values).

Convention:

- Variables $\hat{x}, \hat{y} \dots$ range over arbitrary (i.e., partial) objects.
- Variables $x, y \dots$ range over total objects.

For readability,

$$\forall_x A(x) \quad \text{abbreviates} \quad \forall_{\hat{x}}(T\hat{x} \to A(\hat{x})).$$

Partiality

- Defined functionals D (and hence terms) can be partial.
- Many D's are total (map total arguments into total values).

Convention:

- Variables $\hat{x}, \hat{y} \dots$ range over arbitrary (i.e., partial) objects.
- Variables $x, y \dots$ range over total objects.

For readability,

$$\forall_x A(x) \quad \text{abbreviates} \quad \forall_{\hat{x}}(T\hat{x} \to A(\hat{x})).$$

Partiality

- Defined functionals D (and hence terms) can be partial.
- Many D's are total (map total arguments into total values).

Convention:

- Variables $\hat{x}, \hat{y} \ldots$ range over arbitrary (i.e., partial) objects.
- Variables $x, y \ldots$ range over total objects.

For readability,

$$\forall_x A(x) \quad \text{abbreviates} \quad \forall_{\hat{x}}(T\hat{x} \to A(\hat{x})).$$

Partiality

- Defined functionals D (and hence terms) can be partial.
- Many D's are total (map total arguments into total values).

Convention:

- Variables $\hat{x}, \hat{y} \dots$ range over arbitrary (i.e., partial) objects.
- Variables $x, y \dots$ range over total objects.

For readability,

$$\forall_x A(x) \quad \text{abbreviates} \quad \forall_{\hat{x}}(T\hat{x} \to A(\hat{x})).$$

Partiality

- Defined functionals D (and hence terms) can be partial.
- Many D's are total (map total arguments into total values).

Convention:

- Variables $\hat{x}, \hat{y} \dots$ range over arbitrary (i.e., partial) objects.
- Variables $x, y \dots$ range over total objects.

For readability,

$$\forall_x A(x) \quad \text{abbreviates} \quad \forall_{\hat{x}}(T\hat{x} \to A(\hat{x})).$$

Equality

There are many variants of equality:

- Decidable equality for base types, for instance $=_{\mathbb{N}}$.
- Leibniz equality, inductively defined by the clause $\forall_x (x \equiv x)$.
- Pointwise equality[2]:

$$(f \doteq_{\tau\to\sigma} g) := \forall_{x,y}(x \doteq_\tau y \to fx \doteq_\sigma gy).$$

Extensionality is defined as diagonalization of pointwise equality:

$$(x \in \mathrm{Ext}_\tau) := (x \doteq_\tau x).$$

[2] Robin Gandy, On the axiom of extensionality – Part I, JSL 1956 and Gaisi Takeuti, On a generalized logic calculus, Jap. J. Math. 1953

Equality

There are many variants of equality:

- Decidable equality for base types, for instance $=_{\mathbb{N}}$.
- Leibniz equality, inductively defined by the clause $\forall_x (x \equiv x)$.
- Pointwise equality[2]:

$$(f \doteq_{\tau \to \sigma} g) := \forall_{x,y}(x \doteq_\tau y \to fx \doteq_\sigma gy).$$

Extensionality is defined as diagonalization of pointwise equality:

$$(x \in \mathrm{Ext}_\tau) := (x \doteq_\tau x).$$

[2] Robin Gandy, On the axiom of extensionality – Part I, JSL 1956 and Gaisi Takeuti, On a generalized logic calculus, Jap. J. Math. 1953

Equality

There are many variants of equality:

- Decidable equality for base types, for instance $=_{\mathbb{N}}$.
- Leibniz equality, inductively defined by the clause $\forall_x (x \equiv x)$.
- Pointwise equality[2]:

$$(f \doteq_{\tau\to\sigma} g) := \forall_{x,y}(x \doteq_\tau y \to fx \doteq_\sigma gy).$$

Extensionality is defined as diagonalization of pointwise equality:

$$(x \in \mathrm{Ext}_\tau) := (x \doteq_\tau x).$$

[2] Robin Gandy, On the axiom of extensionality – Part I, JSL 1956 and Gaisi Takeuti, On a generalized logic calculus, Jap. J. Math. 1953

Equality

There are many variants of equality:

- Decidable equality for base types, for instance $=_{\mathbb{N}}$.
- Leibniz equality, inductively defined by the clause $\forall_x (x \equiv x)$.
- Pointwise equality[2]:

$$(f \doteq_{\tau \to \sigma} g) := \forall_{x,y}(x \doteq_\tau y \to fx \doteq_\sigma gy).$$

Extensionality is defined as diagonalization of pointwise equality:

$$(x \in \mathrm{Ext}_\tau) := (x \doteq_\tau x).$$

[2] Robin Gandy, On the axiom of extensionality – Part I, JSL 1956 and Gaisi Takeuti, On a generalized logic calculus, Jap. J. Math. 1953

Equality

There are many variants of equality:

- Decidable equality for base types, for instance $=_{\mathbb{N}}$.
- Leibniz equality, inductively defined by the clause $\forall_x(x \equiv x)$.
- Pointwise equality[2]:

$$(f \doteq_{\tau\to\sigma} g) := \forall_{x,y}(x \doteq_\tau y \to fx \doteq_\sigma gy).$$

Extensionality is defined as diagonalization of pointwise equality:

$$(x \in \mathrm{Ext}_\tau) := (x \doteq_\tau x).$$

[2] Robin Gandy, On the axiom of extensionality – Part I, JSL 1956 and Gaisi Takeuti, On a generalized logic calculus, Jap. J. Math. 1953

Properties

- Ext_τ and ${}^{\mathrm{co}}T_\tau$ are equivalent for closed types of level ≤ 1.
- For every closed type τ the relation $\doteq_\tau$ is an equivalence relation on Ext_τ.
- For every term $t(\vec{x})$ with extensional constants and free variables among $\vec{x}$ we have

$$\vec{x} \doteq_{\vec{\rho}} \vec{y} \to t(\vec{x}) \doteq_\tau t(\vec{y}),$$
$$\vec{x} \in \mathrm{Ext}_{\vec{\rho}} \to t(\vec{x}) \in \mathrm{Ext}_\tau.$$

Properties

- Ext_τ and ${}^{\mathrm{co}}T_\tau$ are equivalent for closed types of level ≤ 1.
- For every closed type τ the relation $\doteq_\tau$ is an equivalence relation on Ext_τ.
- For every term $t(\vec{x})$ with extensional constants and free variables among $\vec{x}$ we have

$$\vec{x} \doteq_{\vec{\rho}} \vec{y} \to t(\vec{x}) \doteq_\tau t(\vec{y}),$$
$$\vec{x} \in \mathrm{Ext}_{\vec{\rho}} \to t(\vec{x}) \in \mathrm{Ext}_\tau.$$

Properties

- Ext_τ and ${}^{\mathrm{co}}T_\tau$ are equivalent for closed types of level ≤ 1.
- For every closed type τ the relation $\doteq_\tau$ is an equivalence relation on Ext_τ.
- For every term $t(\vec{x})$ with extensional constants and free variables among $\vec{x}$ we have

$$\vec{x} \doteq_{\vec{\rho}} \vec{y} \to t(\vec{x}) \doteq_\tau t(\vec{y}),$$

$$\vec{x} \in \mathrm{Ext}_{\vec{\rho}} \to t(\vec{x}) \in \mathrm{Ext}_\tau .$$

Properties

- Ext_τ and ${}^{\mathrm{co}}T_\tau$ are equivalent for closed types of level ≤ 1.
- For every closed type τ the relation $\doteq_\tau$ is an equivalence relation on Ext_τ.
- For every term $t(\vec{x})$ with extensional constants and free variables among $\vec{x}$ we have

$$\vec{x} \doteq_{\vec{\rho}} \vec{y} \rightarrow t(\vec{x}) \doteq_\tau t(\vec{y}),$$
$$\vec{x} \in \mathrm{Ext}_{\vec{\rho}} \rightarrow t(\vec{x}) \in \mathrm{Ext}_\tau.$$

c.r. and n.c.

We have two sorts of inductive predicates and predicate variables,

- "computationally relevant" ones I^{c}, X^{c} and
- "non-computational" ones I^{nc}, X^{nc}.
- We use I, X for both.

This leads to a distinction between c.r. and n.c. formulas.

It allows to "fine tune" the computational content of a proof.

c.r. and n.c.

We have two sorts of inductive predicates and predicate variables,

- "computationally relevant" ones I^{c}, X^{c} and
- "non-computational" ones I^{nc}, X^{nc}.
- We use I, X for both.

This leads to a distinction between c.r. and n.c. formulas.

It allows to "fine tune" the computational content of a proof.

c.r. and n.c.

We have two sorts of inductive predicates and predicate variables,

- "computationally relevant" ones I^{c}, X^{c} and
- "non-computational" ones I^{nc}, X^{nc}.
- We use I, X for both.

This leads to a distinction between c.r. and n.c. formulas.

It allows to "fine tune" the computational content of a proof.

c.r. and n.c.

We have two sorts of inductive predicates and predicate variables,

- "computationally relevant" ones I^{c}, X^{c} and
- "non-computational" ones I^{nc}, X^{nc}.
- We use I, X for both.

This leads to a distinction between c.r. and n.c. formulas.

It allows to "fine tune" the computational content of a proof.

c.r. and n.c.

We have two sorts of inductive predicates and predicate variables,

- "computationally relevant" ones I^{c}, X^{c} and
- "non-computational" ones I^{nc}, X^{nc}.
- We use I, X for both.

This leads to a distinction between c.r. and n.c. formulas.

It allows to "fine tune" the computational content of a proof.

Need "realizability extensions" of c.r. predicates and formulas:

- Assume that we have a global assignment giving for every c.r. predicate variable X of arity $\vec{\rho}$ an n.c. predicate variable $X^{\mathbf{r}}$ of arity $(\vec{\rho}, \xi)$ where ξ is the type variable associated with X.
- We introduce $I^{\mathbf{r}}/{}^{\mathrm{co}}I^{\mathbf{r}}$ for c.r. (co)inductive predicates $I/{}^{\mathrm{co}}I$, e.g.,

$$\mathrm{Even}^{\mathbf{r}}00 \qquad \mathrm{Even}^{\mathbf{r}}nm \to \mathrm{Even}^{\mathbf{r}}(S(Sn))(Sm).$$

- A predicate or formula C is **r**-free if it does not contain any of these $X^{\mathbf{r}}$, $I^{\mathbf{r}}$ or ${}^{\mathrm{co}}I^{\mathbf{r}}$.
- A derivation M is **r**-free if it contains **r**-free formulas only.

Need "realizability extensions" of c.r. predicates and formulas:

- Assume that we have a global assignment giving for every c.r. predicate variable X of arity $\vec{\rho}$ an n.c. predicate variable $X^{\mathbf{r}}$ of arity $(\vec{\rho}, \xi)$ where ξ is the type variable associated with X.
- We introduce $I^{\mathbf{r}}/{}^{\mathrm{co}}I^{\mathbf{r}}$ for c.r. (co)inductive predicates $I/{}^{\mathrm{co}}I$, e.g.,

$$\mathrm{Even}^{\mathbf{r}}00 \qquad \mathrm{Even}^{\mathbf{r}}nm \to \mathrm{Even}^{\mathbf{r}}(S(Sn))(Sm).$$

- A predicate or formula C is **r**-free if it does not contain any of these $X^{\mathbf{r}}$, $I^{\mathbf{r}}$ or ${}^{\mathrm{co}}I^{\mathbf{r}}$.
- A derivation M is **r**-free if it contains **r**-free formulas only.

Need "realizability extensions" of c.r. predicates and formulas:

- Assume that we have a global assignment giving for every c.r. predicate variable X of arity $\vec{\rho}$ an n.c. predicate variable $X^{\mathbf{r}}$ of arity $(\vec{\rho}, \xi)$ where ξ is the type variable associated with X.
- We introduce $I^{\mathbf{r}}/{}^{\mathrm{co}}I^{\mathbf{r}}$ for c.r. (co)inductive predicates $I/{}^{\mathrm{co}}I$, e.g.,

$$\mathrm{Even}^{\mathbf{r}}00 \qquad \mathrm{Even}^{\mathbf{r}}nm \to \mathrm{Even}^{\mathbf{r}}(S(Sn))(Sm).$$

- A predicate or formula C is **r**-free if it does not contain any of these $X^{\mathbf{r}}$, $I^{\mathbf{r}}$ or ${}^{\mathrm{co}}I^{\mathbf{r}}$.
- A derivation M is **r**-free if it contains **r**-free formulas only.

Need "realizability extensions" of c.r. predicates and formulas:

- Assume that we have a global assignment giving for every c.r. predicate variable X of arity $\vec{\rho}$ an n.c. predicate variable $X^{\mathbf{r}}$ of arity $(\vec{\rho}, \xi)$ where ξ is the type variable associated with X.
- We introduce $I^{\mathbf{r}}/{}^{\mathrm{co}}I^{\mathbf{r}}$ for c.r. (co)inductive predicates $I/{}^{\mathrm{co}}I$, e.g.,

$$\mathrm{Even}^{\mathbf{r}}00 \qquad \mathrm{Even}^{\mathbf{r}}nm \to \mathrm{Even}^{\mathbf{r}}(S(Sn))(Sm).$$

- A predicate or formula C is **r**-free if it does not contain any of these $X^{\mathbf{r}}$, $I^{\mathbf{r}}$ or ${}^{\mathrm{co}}I^{\mathbf{r}}$.
- A derivation M is **r**-free if it contains **r**-free formulas only.

Need "realizability extensions" of c.r. predicates and formulas:

- Assume that we have a global assignment giving for every c.r. predicate variable X of arity $\vec{\rho}$ an n.c. predicate variable $X^{\mathbf{r}}$ of arity $(\vec{\rho}, \xi)$ where ξ is the type variable associated with X.
- We introduce $I^{\mathbf{r}}/{}^{\mathrm{co}}I^{\mathbf{r}}$ for c.r. (co)inductive predicates $I/{}^{\mathrm{co}}I$, e.g.,

$$\mathrm{Even}^{\mathbf{r}}00 \qquad \mathrm{Even}^{\mathbf{r}}nm \to \mathrm{Even}^{\mathbf{r}}(S(Sn))(Sm).$$

- A predicate or formula C is **r**-free if it does not contain any of these $X^{\mathbf{r}}$, $I^{\mathbf{r}}$ or ${}^{\mathrm{co}}I^{\mathbf{r}}$.
- A derivation M is **r**-free if it contains **r**-free formulas only.

Definition ($C^{\mathbf{r}}$ for $\mathbf{r}$-free c.r. formulas C)

Let $z \mathbf{r} C$ mean $C^{\mathbf{r}} z$.

$$z \mathbf{r} P\vec{t} := P^{\mathbf{r}}\vec{t}z,$$

$$z \mathbf{r} (A \to B) := \begin{cases} \forall_w (w \mathbf{r} A \to zw \mathbf{r} B) & \text{if } A \text{ is c.r.} \\ A \to z \mathbf{r} B & \text{if } A \text{ is n.c.} \end{cases}$$

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

Definition (Extracted term for an **r**-free proof M of a c.r. A)

$$
\begin{aligned}
\mathrm{et}(u^A) &:= z_u^{\tau(A)} \quad (z_u^{\tau(A)} \text{ uniquely associated to } u^A),\\
\mathrm{et}((\lambda_{u^A} M^B)^{A\to B}) &:= \begin{cases} \lambda_{z_u}\mathrm{et}(M) & \text{if } A \text{ is c.r.}\\ \mathrm{et}(M) & \text{if } A \text{ is n.c.,}\end{cases}\\
\mathrm{et}((M^{A\to B} N^A)^B) &:= \begin{cases} \mathrm{et}(M)\mathrm{et}(N) & \text{if } A \text{ is c.r.}\\ \mathrm{et}(M) & \text{if } A \text{ is n.c.,}\end{cases}\\
\mathrm{et}((\lambda_x M^A)^{\forall_x A}) &:= \mathrm{et}(M),\\
\mathrm{et}((M^{\forall_x A(x)} t)^{A(t)}) &:= \mathrm{et}(M).
\end{aligned}
$$

Definition (Extracted term for an **r**-free proof M of a c.r. A)

$$
\begin{aligned}
\mathrm{et}(u^A) &:= z_u^{\tau(A)} \quad (z_u^{\tau(A)} \text{ uniquely associated to } u^A),\\
\mathrm{et}((\lambda_{u^A} M^B)^{A \to B}) &:= \begin{cases} \lambda_{z_u}\mathrm{et}(M) & \text{if } A \text{ is c.r.} \\ \mathrm{et}(M) & \text{if } A \text{ is n.c.,} \end{cases}\\
\mathrm{et}((M^{A \to B} N^A)^B) &:= \begin{cases} \mathrm{et}(M)\mathrm{et}(N) & \text{if } A \text{ is c.r.} \\ \mathrm{et}(M) & \text{if } A \text{ is n.c.,} \end{cases}\\
\mathrm{et}((\lambda_x M^A)^{\forall_x A}) &:= \mathrm{et}(M),\\
\mathrm{et}((M^{\forall_x A(x)} t)^{A(t)}) &:= \mathrm{et}(M).
\end{aligned}
$$

Definition (Extracted term for an **r**-free proof M of a c.r. A)

$$\mathrm{et}(u^A) := z_u^{\tau(A)} \quad (z_u^{\tau(A)} \text{ uniquely associated to } u^A),$$

$$\mathrm{et}((\lambda_{u^A} M^B)^{A \to B}) := \begin{cases} \lambda_{z_u} \mathrm{et}(M) & \text{if } A \text{ is c.r.} \\ \mathrm{et}(M) & \text{if } A \text{ is n.c.,} \end{cases}$$

$$\mathrm{et}((M^{A \to B} N^A)^B) := \begin{cases} \mathrm{et}(M)\mathrm{et}(N) & \text{if } A \text{ is c.r.} \\ \mathrm{et}(M) & \text{if } A \text{ is n.c.,} \end{cases}$$

$$\mathrm{et}((\lambda_x M^A)^{\forall_x A}) := \mathrm{et}(M),$$

$$\mathrm{et}((M^{\forall_x A(x)} t)^{A(t)}) := \mathrm{et}(M).$$

Definition (Extracted term for an **r**-free proof M of a c.r. A)

$$\mathrm{et}(u^A) := z_u^{\tau(A)} \quad (z_u^{\tau(A)} \text{ uniquely associated to } u^A),$$

$$\mathrm{et}((\lambda_{u^A} M^B)^{A\to B}) := \begin{cases} \lambda_{z_u}\mathrm{et}(M) & \text{if } A \text{ is c.r.} \\ \mathrm{et}(M) & \text{if } A \text{ is n.c.,} \end{cases}$$

$$\mathrm{et}((M^{A\to B} N^A)^B) := \begin{cases} \mathrm{et}(M)\mathrm{et}(N) & \text{if } A \text{ is c.r.} \\ \mathrm{et}(M) & \text{if } A \text{ is n.c.,} \end{cases}$$

$$\mathrm{et}((\lambda_x M^A)^{\forall_x A}) := \mathrm{et}(M),$$

$$\mathrm{et}((M^{\forall_x A(x)} t)^{A(t)}) := \mathrm{et}(M).$$

Definition (Extracted term for an **r**-free proof M of a c.r. A)

$$
\begin{aligned}
\mathrm{et}(u^A) &:= z_u^{\tau(A)} \quad (z_u^{\tau(A)} \text{ uniquely associated to } u^A),\\
\mathrm{et}((\lambda_{u^A} M^B)^{A\to B}) &:= \begin{cases} \lambda_{z_u}\mathrm{et}(M) & \text{if } A \text{ is c.r.}\\ \mathrm{et}(M) & \text{if } A \text{ is n.c.,}\end{cases}\\
\mathrm{et}((M^{A\to B} N^A)^B) &:= \begin{cases} \mathrm{et}(M)\mathrm{et}(N) & \text{if } A \text{ is c.r.}\\ \mathrm{et}(M) & \text{if } A \text{ is n.c.,}\end{cases}\\
\mathrm{et}((\lambda_x M^A)^{\forall_x A}) &:= \mathrm{et}(M),\\
\mathrm{et}((M^{\forall_x A(x)} t)^{A(t)}) &:= \mathrm{et}(M).
\end{aligned}
$$

It remains to define extracted terms for the axioms. Consider a (c.r.) inductively defined predicate I.

- $\mathrm{et}(I_i^+) := \mathrm{C}_i$ and $\mathrm{et}(I^-) := \mathcal{R}$, where the constructor C_i and the recursion operator $\mathcal{R}$ refer to ι_I associated with I.
- $\mathrm{et}({}^{\mathrm{co}}I^-) := \mathrm{D}$ and $\mathrm{et}({}^{\mathrm{co}}I_i^+) := {}^{\mathrm{co}}\mathcal{R}$, where the destructor D and the corecursion operator ${}^{\mathrm{co}}\mathcal{R}$ refer to ι_I again.

It remains to define extracted terms for the axioms. Consider a (c.r.) inductively defined predicate I.

- $\mathrm{et}(I_i^+) := \mathrm{C}_i$ and $\mathrm{et}(I^-) := \mathcal{R}$, where the constructor C_i and the recursion operator $\mathcal{R}$ refer to ι_I associated with I.
- $\mathrm{et}({}^{\mathrm{co}}I^-) := \mathrm{D}$ and $\mathrm{et}({}^{\mathrm{co}}I_i^+) := {}^{\mathrm{co}}\mathcal{R}$, where the destructor D and the corecursion operator ${}^{\mathrm{co}}\mathcal{R}$ refer to ι_I again.

It remains to define extracted terms for the axioms. Consider a (c.r.) inductively defined predicate I.

- $\mathrm{et}(I_i^+) := \mathrm{C}_i$ and $\mathrm{et}(I^-) := \mathcal{R}$, where the constructor C_i and the recursion operator $\mathcal{R}$ refer to ι_I associated with I.
- $\mathrm{et}({}^{\mathrm{co}}I^-) := \mathrm{D}$ and $\mathrm{et}({}^{\mathrm{co}}I_i^+) := {}^{\mathrm{co}}\mathcal{R}$, where the destructor D and the corecursion operator ${}^{\mathrm{co}}\mathcal{R}$ refer to ι_I again.

Theorem (Soundness)

Let M be an $\mathbf{r}$*-free derivation of a formula A from assumptions $u_i \colon C_i$ ($i < n$). Then we can derive*

$$\begin{cases} \mathrm{et}(M)\ \mathbf{r}\ A & \text{if } A \text{ is c.r.} \\ A & \text{if } A \text{ is n.c.} \end{cases}$$

from assumptions

$$\begin{cases} z_{u_i}\ \mathbf{r}\ C_i & \text{if } C_i \text{ is c.r.} \\ C_i & \text{if } C_i \text{ is n.c.} \end{cases}$$

We express

- Kolmogorov's view of "formulas as problems"[3]
- Feferman's dictum "to assert is to realize"[4]

by invariance axioms:

For $\mathbf{r}$-free c.r. formulas A we require as axioms

$$\mathrm{InvAll}_A \colon \forall_z (z \mathbf{r} A \to A),$$
$$\mathrm{InvEx}_A \colon A \to \exists_z (z \mathbf{r} A).$$

Invariance axioms are used in the proof of the soundness theorem.

[3] Zur Deutung der intuitionistischen Logik, Math. Zeitschr., 1932

[4] Constructive theories of functions and classes, Logic Colloquium 78, p.208

We express

- Kolmogorov's view of "formulas as problems"[3]
- Feferman's dictum "to assert is to realize"[4]

by invariance axioms:

For $\mathbf{r}$-free c.r. formulas A we require as axioms

$$\mathrm{InvAll}_A \colon \forall_z (z \mathrel{\mathbf{r}} A \to A),$$
$$\mathrm{InvEx}_A \colon A \to \exists_z (z \mathrel{\mathbf{r}} A).$$

Invariance axioms are used in the proof of the soundness theorem.

[3] Zur Deutung der intuitionistischen Logik, Math. Zeitschr., 1932

[4] Constructive theories of functions and classes, Logic Colloquium 78, p.208

We express

- Kolmogorov's view of "formulas as problems"[3]
- Feferman's dictum "to assert is to realize"[4]

by invariance axioms:

For $\mathbf{r}$-free c.r. formulas A we require as axioms

$$\mathrm{InvAll}_A \colon \forall_z (z \mathbf{\ r\ } A \to A),$$
$$\mathrm{InvEx}_A \colon A \to \exists_z (z \mathbf{\ r\ } A).$$

Invariance axioms are used in the proof of the soundness theorem.

[3] Zur Deutung der intuitionistischen Logik, Math. Zeitschr., 1932
[4] Constructive theories of functions and classes, Logic Colloquium 78, p.208

We express

- Kolmogorov's view of "formulas as problems"[3]
- Feferman's dictum "to assert is to realize"[4]

by invariance axioms:

For **r**-free c.r. formulas A we require as axioms

$$\mathrm{InvAll}_A \colon \forall_z (z \; \mathbf{r} \; A \to A),$$
$$\mathrm{InvEx}_A \colon A \to \exists_z (z \; \mathbf{r} \; A).$$

Invariance axioms are used in the proof of the soundness theorem.

[3] Zur Deutung der intuitionistischen Logik, Math. Zeitschr., 1932

[4] Constructive theories of functions and classes, Logic Colloquium 78, p.208

We express

- Kolmogorov's view of "formulas as problems"[3]
- Feferman's dictum "to assert is to realize"[4]

by invariance axioms:

For **r**-free c.r. formulas A we require as axioms

$$\mathrm{InvAll}_A\colon \forall_z (z \mathbf{\ r\ } A \to A),$$
$$\mathrm{InvEx}_A\colon A \to \exists_z (z \mathbf{\ r\ } A).$$

Invariance axioms are used in the proof of the soundness theorem.

[3] Zur Deutung der intuitionistischen Logik, Math. Zeitschr., 1932

[4] Constructive theories of functions and classes, Logic Colloquium 78, p.208

Real numbers

- Real numbers are given as Cauchy sequences of rationals with an explicitly given modulus.

```
;; ApproxSplitBoole
(set-goal "all x1,x2,x3,p(Real x1 -> Real x2 -> Real x3 ->
  RealLt x1 x2 p -> exl boole(
  (boole -> x3<<=x2) andi ((boole -> F) -> x1<<=x3)))")
```

Continuous functions

- Continuous functions on the reals are determined by their values on rationals.
- On closed intervals they come with a modulus of uniform continuity.

Continuous functions

- Continuous functions on the reals are determined by their values on rationals.
- On closed intervals they come with a modulus of uniform continuity.

IVTAux

Let $f\colon I \to \mathbb{R}$ be continuous, with a uniform modulus q of increase.
Let $a < b$ be rationals in I such that

$$a \leq c < d \leq b \quad \text{and} \quad f(c) \leq 0 \leq f(d).$$

Then we can construct c_1, d_1 with

$$d_1 - c_1 = \frac{1}{2}(d - c),$$

such that again

$$a \leq c \leq c_1 < d_1 \leq d \leq b \quad \text{and} \quad f(c_1) \leq 0 \leq f(d_1).$$

IVTAux

Let $f : I \to \mathbb{R}$ be continuous, with a uniform modulus q of increase. Let $a < b$ be rationals in I such that

$$a \leq c < d \leq b \quad \text{and} \quad f(c) \leq 0 \leq f(d).$$

Then we can construct c_1, d_1 with

$$d_1 - c_1 = \frac{1}{2}(d - c),$$

such that again

$$a \leq c \leq c_1 < d_1 \leq d \leq b \quad \text{and} \quad f(c_1) \leq 0 \leq f(d_1).$$

Proof.

Let $b_0 = c$ and $b_{n+1} = b_n + \frac{1}{4}(d - c)$ for $n \leq 3$, hence $b_4 = d$.

From $\frac{1}{2^p} < d - c$ we obtain $\frac{1}{2^{p+2}} \leq b_{n+1} - b_n$, hence $f(b_n) <_{p+2+q} f(b_{n+1})$.

- First compare 0 with $f(b_1) < f(b_2)$, using ApproxSplit.
- In case $0 \leq f(b_2)$ let $c_1 = b_0 = c$ and $d_1 = b_2$.
- In case $f(b_1) \leq 0$ compare 0 with $f(b_2) < f(b_3)$, using ApproxSplit again.
- In case $0 \leq f(b_3)$ let $c_1 = b_1$ and $d_1 = b_3$.
- In case $f(b_2) \leq 0$ let $c_1 = b_2$ and $d_1 = b_4 = d$. □

Proof.

Let $b_0 = c$ and $b_{n+1} = b_n + \frac{1}{4}(d - c)$ for $n \leq 3$, hence $b_4 = d$.
From $\frac{1}{2^p} < d - c$ we obtain $\frac{1}{2^{p+2}} \leq b_{n+1} - b_n$, hence
$f(b_n) <_{p+2+q} f(b_{n+1})$.

- First compare 0 with $f(b_1) < f(b_2)$, using ApproxSplit.
- In case $0 \leq f(b_2)$ let $c_1 = b_0 = c$ and $d_1 = b_2$.
- In case $f(b_1) \leq 0$ compare 0 with $f(b_2) < f(b_3)$, using ApproxSplit again.
- In case $0 \leq f(b_3)$ let $c_1 = b_1$ and $d_1 = b_3$.
- In case $f(b_2) \leq 0$ let $c_1 = b_2$ and $d_1 = b_4 = d$. □

Proof.

Let $b_0 = c$ and $b_{n+1} = b_n + \frac{1}{4}(d - c)$ for $n \leq 3$, hence $b_4 = d$.
From $\frac{1}{2^p} < d - c$ we obtain $\frac{1}{2^{p+2}} \leq b_{n+1} - b_n$, hence
$f(b_n) <_{p+2+q} f(b_{n+1})$.

- First compare 0 with $f(b_1) < f(b_2)$, using ApproxSplit.
- In case $0 \leq f(b_2)$ let $c_1 = b_0 = c$ and $d_1 = b_2$.
- In case $f(b_1) \leq 0$ compare 0 with $f(b_2) < f(b_3)$, using ApproxSplit again.
- In case $0 \leq f(b_3)$ let $c_1 = b_1$ and $d_1 = b_3$.
- In case $f(b_2) \leq 0$ let $c_1 = b_2$ and $d_1 = b_4 = d$. □

Proof.

Let $b_0 = c$ and $b_{n+1} = b_n + \frac{1}{4}(d - c)$ for $n \leq 3$, hence $b_4 = d$. From $\frac{1}{2^p} < d - c$ we obtain $\frac{1}{2^{p+2}} \leq b_{n+1} - b_n$, hence $f(b_n) <_{p+2+q} f(b_{n+1})$.

- First compare 0 with $f(b_1) < f(b_2)$, using ApproxSplit.
- In case $0 \leq f(b_2)$ let $c_1 = b_0 = c$ and $d_1 = b_2$.
- In case $f(b_1) \leq 0$ compare 0 with $f(b_2) < f(b_3)$, using ApproxSplit again.
- In case $0 \leq f(b_3)$ let $c_1 = b_1$ and $d_1 = b_3$.
- In case $f(b_2) \leq 0$ let $c_1 = b_2$ and $d_1 = b_4 = d$. □

Proof.

Let $b_0 = c$ and $b_{n+1} = b_n + \frac{1}{4}(d - c)$ for $n \leq 3$, hence $b_4 = d$. From $\frac{1}{2^p} < d - c$ we obtain $\frac{1}{2^{p+2}} \leq b_{n+1} - b_n$, hence $f(b_n) <_{p+2+q} f(b_{n+1})$.

- First compare 0 with $f(b_1) < f(b_2)$, using ApproxSplit.
- In case $0 \leq f(b_2)$ let $c_1 = b_0 = c$ and $d_1 = b_2$.
- In case $f(b_1) \leq 0$ compare 0 with $f(b_2) < f(b_3)$, using ApproxSplit again.
- In case $0 \leq f(b_3)$ let $c_1 = b_1$ and $d_1 = b_3$.
- In case $f(b_2) \leq 0$ let $c_1 = b_2$ and $d_1 = b_4 = d$. □

Proof.

Let $b_0 = c$ and $b_{n+1} = b_n + \frac{1}{4}(d - c)$ for $n \leq 3$, hence $b_4 = d$. From $\frac{1}{2^p} < d - c$ we obtain $\frac{1}{2^{p+2}} \leq b_{n+1} - b_n$, hence $f(b_n) <_{p+2+q} f(b_{n+1})$.

- First compare 0 with $f(b_1) < f(b_2)$, using ApproxSplit.
- In case $0 \leq f(b_2)$ let $c_1 = b_0 = c$ and $d_1 = b_2$.
- In case $f(b_1) \leq 0$ compare 0 with $f(b_2) < f(b_3)$, using ApproxSplit again.
- In case $0 \leq f(b_3)$ let $c_1 = b_1$ and $d_1 = b_3$.
- In case $f(b_2) \leq 0$ let $c_1 = b_2$ and $d_1 = b_4 = d$. □

Proof.

Let $b_0 = c$ and $b_{n+1} = b_n + \frac{1}{4}(d - c)$ for $n \leq 3$, hence $b_4 = d$. From $\frac{1}{2^p} < d - c$ we obtain $\frac{1}{2^{p+2}} \leq b_{n+1} - b_n$, hence $f(b_n) <_{p+2+q} f(b_{n+1})$.

- First compare 0 with $f(b_1) < f(b_2)$, using ApproxSplit.
- In case $0 \leq f(b_2)$ let $c_1 = b_0 = c$ and $d_1 = b_2$.
- In case $f(b_1) \leq 0$ compare 0 with $f(b_2) < f(b_3)$, using ApproxSplit again.
- In case $0 \leq f(b_3)$ let $c_1 = b_1$ and $d_1 = b_3$.
- In case $f(b_2) \leq 0$ let $c_1 = b_2$ and $d_1 = b_4 = d$. □

IVT

Let $f: I \to \mathbb{R}$ be continuous, with a uniform modulus of increase. Let $a < b$ be rational numbers in I such that $f(a) \leq 0 \leq f(b)$. Then we can find $x \in [a, b]$ such that $f(x) = 0$.

Proof.

Iterating the construction in IVTAux, we construct two sequences $(c_n)_n$ and $(d_n)_n$ of rationals such that for all n

$$a = c_0 \leq c_1 \leq \cdots \leq c_n < d_n \leq \cdots \leq d_1 \leq d_0 = b,$$
$$f(c_n) \leq 0 \leq f(d_n),$$
$$d_n - c_n = \frac{1}{2^n}(b - a).$$

Let x, y be given by the Cauchy sequences $(c_n)_n$ and $(d_n)_n$ with the obvious modulus. As f is continuous, $f(x) = 0 = f(y)$ for the real number $x = y$. □

IVT

Let $f\colon I \to \mathbb{R}$ be continuous, with a uniform modulus of increase. Let $a < b$ be rational numbers in I such that $f(a) \leq 0 \leq f(b)$. Then we can find $x \in [a, b]$ such that $f(x) = 0$.

Proof.

Iterating the construction in IVTAux, we construct two sequences $(c_n)_n$ and $(d_n)_n$ of rationals such that for all n

$$
\begin{aligned}
&a = c_0 \leq c_1 \leq \cdots \leq c_n < d_n \leq \cdots \leq d_1 \leq d_0 = b,\\
&f(c_n) \leq 0 \leq f(d_n),\\
&d_n - c_n = \frac{1}{2^n}(b - a).
\end{aligned}
$$

Let x, y be given by the Cauchy sequences $(c_n)_n$ and $(d_n)_n$ with the obvious modulus. As f is continuous, $f(x) = 0 = f(y)$ for the real number $x = y$. □

IVT

Let $f : I \to \mathbb{R}$ be continuous, with a uniform modulus of increase. Let $a < b$ be rational numbers in I such that $f(a) \leq 0 \leq f(b)$. Then we can find $x \in [a, b]$ such that $f(x) = 0$.

Proof.

Iterating the construction in IVTAux, we construct two sequences $(c_n)_n$ and $(d_n)_n$ of rationals such that for all n

$$
\begin{aligned}
& a = c_0 \leq c_1 \leq \cdots \leq c_n < d_n \leq \cdots \leq d_1 \leq d_0 = b, \\
& f(c_n) \leq 0 \leq f(d_n), \\
& d_n - c_n = \frac{1}{2^n}(b - a).
\end{aligned}
$$

Let x, y be given by the Cauchy sequences $(c_n)_n$ and $(d_n)_n$ with the obvious modulus. As f is continuous, $f(x) = 0 = f(y)$ for the real number $x = y$. □

Example of a continuous function

We represent the continuous real function $x^2 - 2$ on $[1, 2]$ by its values on the rationals:

```
(add-program-constant "SqRtTwo" (py "cont"))
(add-computation-rules
 "SqRtTwo"
 "ContConstr 1 2([a,n]a*a-2)([p]Zero)([p]p+3)~1 2")

;; SqRtTwoApprox
(set-goal
 "exr x(Real x andr SqRtTwo x===0 andr
       all r exl c abs(c+ ~x)<<=(1#2**r))")
```

```
(add-sound "SqRtTwoApprox")

;; ok, SqRtTwoApproxSound has been added as a new theorem:
;; ... with computation rule

;; cSqRtTwoApprox eqd
;; cRealApprox
;; (cIVTFinal(ContConstr 1 2([a,n]a*a+IntN 2)
;;            ([p]Zero)([p]p+3)IntN 1 2)1 1)
```

```
(terms-to-haskell-program
 "~/temp/sqrttwo.hs"
 (list (list (pt "cSqRtTwoApprox") "sqrtwo")))

;; $ ghci sqrttwo.hs

;; *Main> cSqRtTwoApprox 50
;; 1592262918131443 % 1125899906842624

(exact->inexact 1592262918131443/1125899906842624)
;; 1.414213562373095

(sqrt 2)
;; 1.4142135623730951
```

At 50 we already have 15 correct decimal digits.

Further aplications in constructive analysis.

- Verified algorithms for arithmetic on stream-represented real numbers.
- Functional equation of the exponential function.
- Verified algorithm to find for a given real x some p such that

$$\frac{1}{2^p} \leq e^x.$$

Further aplications in constructive analysis.

- Verified algorithms for arithmetic on stream-represented real numbers.
- Functional equation of the exponential function.
- Verified algorithm to find for a given real x some p such that

$$\frac{1}{2^p} \leq e^x.$$

Further aplications in constructive analysis.

- Verified algorithms for arithmetic on stream-represented real numbers.
- Functional equation of the exponential function.
- Verified algorithm to find for a given real x some p such that

$$\frac{1}{2^p} \leq e^x.$$

Further aplications in constructive analysis.

- Verified algorithms for arithmetic on stream-represented real numbers.
- Functional equation of the exponential function.
- Verified algorithm to find for a given real x some p such that

$$\frac{1}{2^p} \leq e^x.$$

Conclusion

- In TCF the computational content of a proof M is represented by an extracted term $\mathrm{et}(M)$ in the language of TCF.
- The soundness theorem provides a formal verification in TCF that the extracted term realizes the formula ("specification"). This is automated in Minlog.
- Since extraction ignores n.c. parts of the proof, $\mathrm{et}(M)$ is much shorter than M.
- For efficiency, in a second step one can translate the extracted term to a functional programming language. Minlog does this for Scheme and Haskell.

Conclusion

- In TCF the computational content of a proof M is represented by an extracted term $\mathrm{et}(M)$ in the language of TCF.
- The soundness theorem provides a formal verification in TCF that the extracted term realizes the formula ("specification"). This is automated in Minlog.
- Since extraction ignores n.c. parts of the proof, $\mathrm{et}(M)$ is much shorter than M.
- For efficiency, in a second step one can translate the extracted term to a functional programming language. Minlog does this for Scheme and Haskell.

Conclusion

- In TCF the computational content of a proof M is represented by an extracted term $\mathrm{et}(M)$ in the language of TCF.
- The soundness theorem provides a formal verification in TCF that the extracted term realizes the formula ("specification"). This is automated in Minlog.
- Since extraction ignores n.c. parts of the proof, $\mathrm{et}(M)$ is much shorter than M.
- For efficiency, in a second step one can translate the extracted term to a functional programming language. Minlog does this for Scheme and Haskell.

Conclusion

- In TCF the computational content of a proof M is represented by an extracted term $\mathrm{et}(M)$ in the language of TCF.
- The soundness theorem provides a formal verification in TCF that the extracted term realizes the formula ("specification"). This is automated in Minlog.
- Since extraction ignores n.c. parts of the proof, $\mathrm{et}(M)$ is much shorter than M.
- For efficiency, in a second step one can translate the extracted term to a functional programming language. Minlog does this for Scheme and Haskell.