# A theory of computable functionals

Helmut Schwichtenberg

Mathematisches Institut, LMU, München

FP 2023, Nis, June 2023

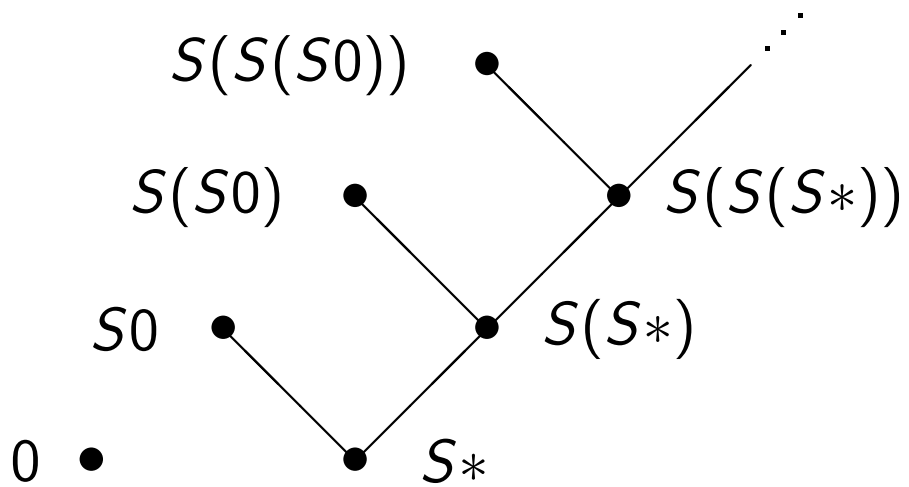
# A theory of computable functionals (TCF)

TCF is similar to $\mathrm{HA}^\omega$, but we

- add inductively and coinductively defined predicates,
- distinguish computationally relevant (c.r.) and non-computational (n.c.) predicates,
- add realizability predicates (internal "meta"-step),
- allow partial functionals, defined by equations (possibly non-terminating, like corecursion),
- use minimal logic: only $\to$, $\forall$ primitive. $\vee$, $\wedge$, $\exists$ inductively defined.

The proof assistant Minlog implements TCF.

- TCF has an intended model: partial continuous functionals.
- Defined via information systems (Scott). Has function spaces.
- It consists of ideals (infinite) approximated by tokens (finite).
- Ideals are consistent and deductively closed sets of tokens.
- Tokens are constructor trees with possibly $*$ at some leaves.
- Examples: natural numbers $\mathbb{N}$, binary trees $\mathbb{Y}$.

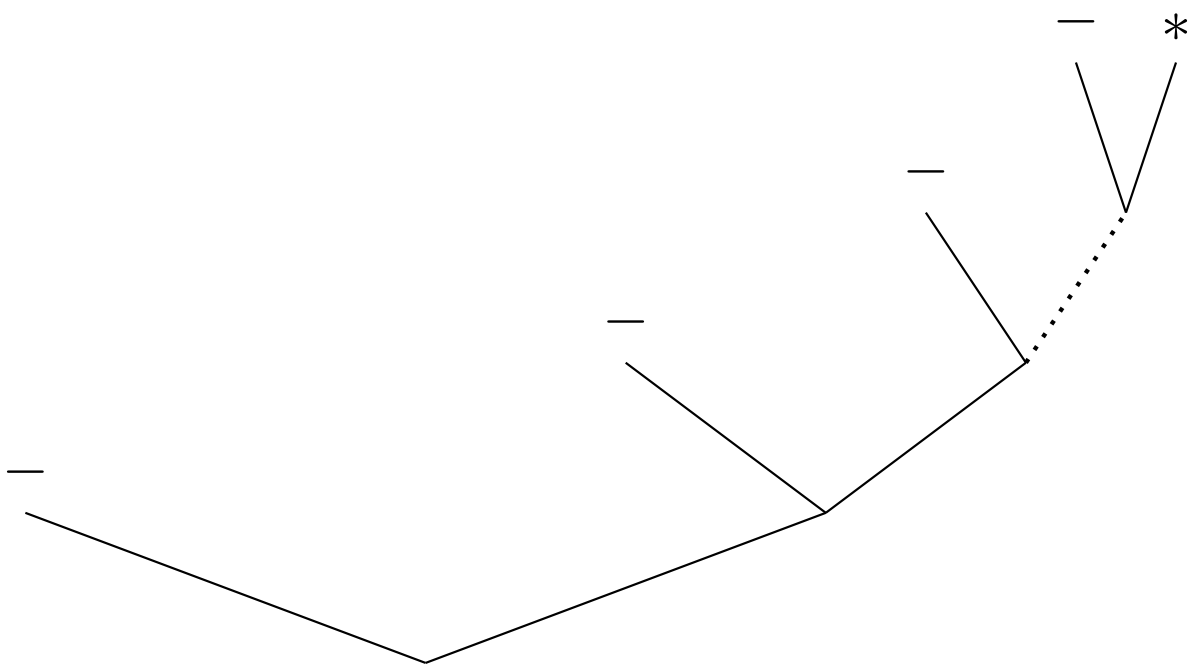



- $\{S0, S(S*)\}$ is inconsistent.
- $\{S*, S(S*)\}$ is an ideal.
- $\{S*, S(S*), S(S0)\}$ is an ideal ("total").
- $\{S*, S(S*), S(S(S*)), \dots\}$ is an infinite ideal ("cototal").

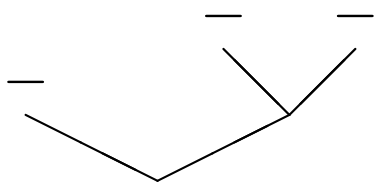
The base type $\mathbb{Y}$ (binary trees) is given by the constructors

$$\begin{aligned} - &: \mathbb{Y} && \text{(leaf)}, \\ \mathrm{C} &: \mathbb{Y} \to \mathbb{Y} \to \mathbb{Y} && \text{(branch)}. \end{aligned}$$

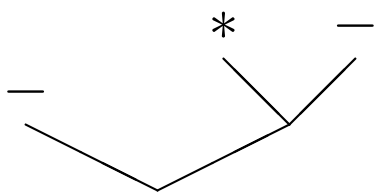
Example of a cototal ideal in $\mathbb{Y}$: all tokens



Example of a total ideal in $\mathbb{Y}$: deductive closure of



Example of a neither total nor cototal ideal: deductive closure of



(Co)inductive predicates. Example. Let $\mathrm{Sd} := \{-1, 0, 1\}$.

$$I_0^+ : 0 \in I, \quad I_1^+ : \forall_{d \in \mathrm{Sd}, x \in I} \frac{x+d}{2} \in I, \qquad \text{clauses}$$

$$I^- : 0 \in X \to \forall_{d \in \mathrm{Sd}, x \in X \cap I} \frac{x+d}{2} \in I \to I \subseteq X \qquad \text{lfp, ind.}$$

$${}^{\mathrm{co}}I^- : x \in {}^{\mathrm{co}}I \to x = 0 \vee \exists_{d \in \mathrm{Sd}, x' \in I}\, x = \frac{x'+d}{2} \qquad \text{closure}$$

$${}^{\mathrm{co}}I^+ : \forall_{x \in X} \left( x = 0 \vee \exists_{d \in \mathrm{Sd}, x' \in X \cup {}^{\mathrm{co}}I}\, x = \frac{x'+d}{2} \right) \to X \subseteq {}^{\mathrm{co}}I \qquad \text{gfp}$$

Greatest-fixed-point (or coinduction) axiom ${}^{\mathrm{co}}I^+$: every "competitor" satisfying the same closure axiom is contained in ${}^{\mathrm{co}}I$.

The n.c. Leibniz equality $\equiv$ is defined by

$$\equiv^+ : x^\tau \equiv x^\tau \qquad \equiv^- : x \equiv y \to \forall_x Xxx \to Xxy$$

We can deduce the property Leibniz used as a definition.

Lemma (Compatibility of $\mathrm{EqD}$)

$$x \equiv y \to A(x) \to A(y).$$

**Proof**: By the lfp-axiom with $X := \{\, x, y \mid A(x) \to A(y) \,\}$.

Using compatibility of $\equiv$ one proves symmetry and transitivity.
Define falsity by $\mathbf{F} := (\mathbb{ff} \equiv \mathbb{tt})$.

Theorem (Ex-falso-quodlibet)

*We can derive* $\mathbf{F} \to A$ *from assumptions* $\mathrm{Ef}_Y : \forall_{\vec{x}}(\mathbf{F} \to Y\vec{x}\,)$ *for predicate variables* $Y$ *strictly positive in* $A$.

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 $\xi$ is the type variable associated with $X$.
- We introduce $I^{\mathbf{r}}/{}^{\mathrm{co}}I^{\mathbf{r}}$ for c.r. (co)inductive predicates $I/{}^{\mathrm{co}}I$:

$$\begin{array}{ll} 0 \in \mathrm{Even} & n \in \mathrm{Even} \to S(Sn) \in \mathrm{Even} \\ \mathrm{Even}^{\mathbf{r}} 00 & \mathrm{Even}^{\mathbf{r}} nm \to \mathrm{Even}^{\mathbf{r}}(S(Sn))(Sm) \end{array}$$

- 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. $A$)

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

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

## Theorem (Soundness)

*Let $M$ be an* $\mathbf{r}$*-free derivation of a formula $A$ from assumptions $u_i : C_i$ $(i < n)$. 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"[1]
- Feferman's dictum "to assert is to realize"[2]

by invariance axioms:

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

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

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

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

- Proofs: on constructive real arithmetic (Bishop).
- Algorithms: operate on reals represented by streams of digits.
- Algorithms via proofs: extract computational content. Needs $P(x)$ and $P^{\mathbf{r}}(x, t)$ "$t$ realizes $P(x)$" for predicates $P$.
- Verification: formal proof that the extracted term realizes $A$.

## Theorem

$$x, y \in {}^{\mathrm{co}}I \to \frac{x+y}{2} \in {}^{\mathrm{co}}I.$$

## Proof.

First show that[3]

$$\left\{ \frac{x+y}{2} \,\middle|\, x, y \in {}^{\mathrm{co}}I \right\} \subseteq \left\{ \frac{x+y+i}{2} \,\middle|\, x, y \in {}^{\mathrm{co}}I, i \in \mathrm{Sd}_2 \right\},$$

$\mathrm{Sd}_2 := \{-2, -1, 0, 1, 2\}$. Then prove that the second set also satisfies the closure axiom for ${}^{\mathrm{co}}I$. Coinduction gives the claim. □

*The computational content of this proof is an algorithm operating on stream representations of reals.*

[3] Berger & Seisenberger, 2010

- Strong language, but controlled existence axioms (Kreisel).
- Functions (other than constructors) can only be defined by computation rules, e.g.,

$$n + 0 = n,$$
$$n + S(m) = S(n + m).$$

  No termination proof is required, hence partial functions.
- Predicates can be defined inductively or coinductively.

- In $\mathrm{TCF}$ the computational content of a proof $M$ is represented by an extracted term $\mathrm{et}(M)$ in the language of $\mathrm{TCF}$.
- 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.
- The Soundness theorem provides a formal verfication in $\mathrm{TCF}$ that the extracted term realizes the formula ("specification"). This is automated in Minlog.