

A theory of computable functionals

Helmut Schwichtenberg

Mathematisches Institut, LMU, München

Verona, 22. March 2024

- 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.

Example

- Disjunction is viewed as a nullary predicate with 2 parameters.
- In $A \vee B$, the computational status of A and B determines a variant of \vee :

$A \vee^d B$ for A c.r. and B c.r. (d for double),

$A \vee^l B$ for A c.r. and B n.c. (l for left),

$A \vee^r B$ for A n.c. and B c.r. (r for right),

$A \vee^u B$ for A n.c. and B n.c. (u for uniform),

and $A \vee^{\text{nc}} B$ (non-computational) for A n.c. and B n.c.

- The “types” in the first 4 cases are

$$\alpha + \beta, \quad \alpha + \mathbb{U}, \quad \mathbb{U} + \beta, \quad \mathbb{B}.$$

Logic

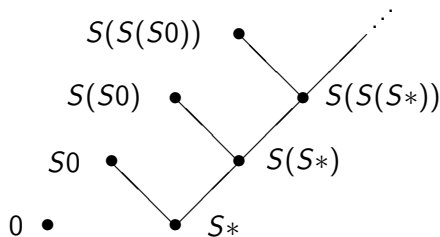
- A constructive **extension** of classical logic, by adding “strong” variants of \forall, \exists to the classical $\tilde{\forall}, \tilde{\exists}$:

$$A \tilde{\forall} B := (\neg A \rightarrow \neg B \rightarrow \perp), \quad \tilde{\exists}_x A := \neg \forall_x \neg A.$$

- 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 look for a theory describing a concrete model,
- Scott-Ershov model of partial continuous functionals¹. 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 $*$.

¹Dana 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").

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:

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

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

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

Recursion $\mathcal{R}_{\mathbb{N}}^\tau: \mathbb{N} \rightarrow \tau \rightarrow (\mathbb{N} \rightarrow \tau \rightarrow \tau) \rightarrow \tau$.

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

Predicates and formulas

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

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

$$0 \in T_{\mathbb{N}}, \quad n \in T_{\mathbb{N}} \rightarrow Sn \in T_{\mathbb{N}}.$$

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

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

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,

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

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 equality²:

$$(f \dot{=}_{\tau \rightarrow \sigma} g) := \forall_{x,y}(x \dot{=}_{\tau} y \rightarrow fx \dot{=}_{\sigma} gy).$$

Extensionality is defined as diagonalization of pointwise equality:

$$(x \in \text{Ext}_{\tau}) := (x \dot{=}_{\tau} x).$$

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

Example of a non-extensional functional

- Define $f, g: \mathbb{N} \rightarrow \mathbb{N}$ by $f\hat{n} = 0$ and $g0 = 0, g(S\hat{n}) = g\hat{n}$.
- Then $f\perp_{\mathbb{N}} = 0$ because of the computation rules for f .
- For $g\perp_{\mathbb{N}}$ no computation rule fits, therefore $\llbracket g\perp_{\mathbb{N}} \rrbracket$ is the empty ideal $\llbracket \perp_{\mathbb{N}} \rrbracket$.
- Hence $f \dot{=} g$, i.e., $\forall_{n,m}(n \dot{=}_{\mathbb{N}} m \rightarrow fn \dot{=}_{\mathbb{N}} gm)$, since $n \dot{=}_{\mathbb{N}} m$ implies both $n \in T_{\mathbb{N}}$ and $n \equiv m$.
- Therefore the functional F defined by $Fh = h\perp_{\mathbb{N}}$ maps the pointwise equal f, g to different values.

Properties

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

$$\begin{aligned}\vec{x} \dot{=}_{\vec{\rho}} \vec{y} &\rightarrow t(\vec{x}) \dot{=}_\tau t(\vec{y}), \\ \vec{x} \in \text{Ext}_{\vec{\rho}} &\rightarrow t(\vec{x}) \in \text{Ext}_\tau.\end{aligned}$$

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^r of arity $(\vec{\rho}, \xi)$ where ξ is the type variable associated with X .
- We introduce $I^r / {}^{co}I^r$ for c.r. (co)inductive predicates $I / {}^{co}I$, e.g.,

$$\text{Even}^r 00 \quad \text{Even}^r nm \rightarrow \text{Even}^r (S(Sn))(Sm).$$

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

Definition (C^r for r -free c.r. formulas C)

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

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

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

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

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

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

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

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

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

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

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

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

Theorem (Soundness)

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

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

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

$$\text{InvAll}_A: \forall_z (z \mathbf{r} A \rightarrow A),$$

$$\text{InvEx}_A: A \rightarrow \exists_z (z \mathbf{r} A).$$

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

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

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

A simple example of program extraction

Let $\text{DGs}: \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$ (for decreasing Gauß sum) defined by

$$\begin{aligned}\text{DGs}(n, 0) &:= 0, \\ \text{DGs}(n, m + 1) &:= \text{DGs}(n, m) + (n - m).\end{aligned}$$

Then $\text{DGs}(n, m) = n + (n - 1) + \cdots + (n - (m - 1))$ with m summands.

Lemma

Every $k < \text{DGs}(n, n)$ can uniquely be written in the form

$$k = \text{DGs}(n, m) + l \quad \text{with } l < n - m.$$

The existence part of the proof is done in such a way that it provides a reasonable algorithm to compute m, l from k (in contrast to a proof based on searching). We split the proof:

$$k + 1 < \text{DGs}(n, m) \rightarrow \exists_{m,l}(k = \text{DGs}(n, m) + l \wedge l < n - m), \quad (1)$$

$$k + 1 = \text{DGs}(n, m) \rightarrow \exists_{m,l}(k = \text{DGs}(n, m) + l \wedge l < n - m). \quad (2)$$

(1) is proved by induction on k , (2) by cases on n .

Extracted term

```
[n,n0]
[if (Succ n0<DGs n n)
  ((Rec nat=>nat yprod nat)n0(0 pair 0)
   ([n1,m1]
    [if m1 ([n2,n3] [if (Succ n3<n--n2)
                        (n2 pair Succ n3)
                        (Succ n2 pair 0)])))]
 [if n (0 pair 0) ([n1]n1 pair 0)]]
```


Applications in constructive analysis.

- Intermediate value theorem.
- 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 $\text{et}(M)$ **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, $\text{et}(M)$ 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.