Proof Theory and Proof Mining I: Computablity and Analysis

Jeremy Avigad

Department of Philosophy and
Department of Mathematical Sciences
Carnegie Mellon University

July 2015

Historical background

Through the nineteenth century, mathematics was essentially computational:

- Euclid's *Elements* was about geometric constructions.
- Algebra was about solving equations.
- Probability was about calculating odds.
- Analysis was about calculating volumes, trajectories, probabilities, etc.

Historical background

This changed in the nineteenth century:

- Gauss gave a nonconstructive proof of the fundamental theorem of algebra.
- Bolzano gave a nonconstructive proof of the Bolzano-Weierstraß theorem.
- Dirichlet considered the "function" $f(x)$ on reals which is 1 if x is rational, 0 if x is irrational.
- Dedekind considered operations on ideals.
- Cantor considered transfinite operations on sets.
- Hilbert proved the *Hilbert basis theorem*: given any sequence $f_1, f_2, f_3, \ldots$ in (say) $\mathbb{Z}[\vec{x}]$, there is an n such that for every $m \geq n$, f_m is in $(f_1, \ldots, f_n)$.
- Dedekind and others emphasized the importance of "conceptual understanding" over calculation.

Historical background

Things came to a head in the early twentieth century, with the "crisis of foundations."

- Set-theoretic paradoxes raised the question of the *consistency* of the new methods.
- A closer look at foundational debates show concerns over the *appropriateness* of abstract methods – cf. Kronecker, Hilbert, the "French analysts" (Baire, Borel, Lebesgue), Brouwer, Weyl, . . .

Hilbert's *Beweistheorie* (*Proof Theory*):

- Narrowly construed as an attempt to establish consistency.
- More generally, evolved into an attempt to reconcile abstract/ideal and concrete/computational views of mathematics, and "reduce" classical methods to finitary or constructive methods.

Siegel to Mordell (1962)

When I first saw [Lang's *Diophantine Geometry*], about a year ago, I was disgusted with the way in which my own contributions to the subject had been disfigured and made unintelligible. . .

The whole style of the author contradicts the sense for simplicity and honesty which we admire in the works of the masters in number theory — Lagrange, Gauss, or on a smaller scale, Hardy, Landau. Just now Lang has published another book on algebraic numbers which, in my opinion, is still worse than the former one. I see a pig broken into a beautiful garden and rooting up all flowers and trees. . .

I am afraid that mathematics will perish before the end of this century if the present trend for senseless abstraction — as I call it: theory of the empty set — cannot be blocked up. . . .

Bishop, *Constructive Analysis* (1967)

It appears . . . that there are certain mathematical statements that are merely evocative, which make assertions without empirical validity. There are also mathematical statements of immediate empirical validity, which say that certain performable operations will produce certain observable results, for instance, the theorem that every positive integer is the sum of four squares. Mathematics is a mixture of the real and the ideal, sometimes one, sometimes the other, often so presented that it is hard to tell which is which. The realistic component of mathematics — the desire for pragmatic interpretation — supplies the control which determines the course of development and keeps mathematics from lapsing into meaningless formalism. The idealistic component permits simplifications and opens possibilities which would otherwise be closed. The methods of proof and objects of investigation have been idealized to form a game, but the actual conduct of the game is ultimately motivated by pragmatic considerations.

Proof Theory and Proof Mining

The idea:

- Represent classical methods in formal axiomatic systems.
- Study such systems with an eye towards "concrete" or "pragmatic" content.

Twentieth century proof theory: "foundational reduction," e.g. classical to constructive.

Contemporary proof mining: try to find quantitative or computational content "hidden" in everyday proofs.

Sequence of Topics

1. Computable Analysis
2. Formal Theories of Analysis
3. The Dialectica Interpretation and Applications
4. Ultraproducts and Nonstandard Analysis

Computable Analysis

Goal: understand the extent to which ordinary theorems of analysis can be understood in computational terms.

This is a foundational study; we will not be directly concerned with *efficient* computation.

Computability

Let f be a partial function from $\mathbb{N} \to \mathbb{N}$. TFAE:

- f is computable by a Turing machine
- f is partial recursive (primitive recursion and μ)
- f is representable in the λ calculus
- f is computable by a register machine
- ...

By the Church-Turing thesis, these capture what it means to say that f is "computable."

Below, by *computable function*, I will mean a total computable function.

Computability

Kleene's T predicate $T(e, x, s)$ says "s is a halting computation sequence for Turing machine e on input x."

The set $\{\langle e, x\rangle \mid \exists s\ T(e, x, s)\}$ is the *halting problem*.

It is computably enumerable but not computable. (In fact, it is a complete computably enumerable set.)

Computable reals

A real number r is *computable* if there is a computable function $\alpha : \mathbb{N} \to \mathbb{Q}$ such that for every i, $|\alpha(i) - r| < 2^{-i}$.

Call such an α a *name*. So r is computable if it has a computable name.

Specker sequences

Theorem (Specker). There is a computable, nondecreasing sequence (a_n) of rationals in $[0, 1]$ with no computable limit.

- In general, one can always compute a name for the limit with the halting problem as an oracle.
- Conversely, there is a sequence (a_n) such that the halting problem is computable from any such name.

Proof. Let $S_n = \{e < n \mid \exists s < n \; T(e, 0, s)\}$.

Let $a_n = \sum_{e \in S_n} 2^{-e}$. If a_n is within 2^{e+1} of the limit and $e < n$ hasn't halted by then, it never will.

Computable functions on $\mathbb{R}$

Definition. A function $f : \mathbb{R} \to \mathbb{R}$ is computable if there is a computable function $F(\alpha, n)$, such that whenever α names a real number x, $\lambda n\, F(\alpha, n)$ represents the real number $f(x)$.

Notes:

- α is given as an *oracle*.
- Only a finite portion of α is read by the computation.

An equivalent characterization

Let $\mathcal{I}$ be the set of (codes of) open intervals (a, b) with rational endpoints, and for $I \in \mathcal{I}$ let $|I|$ be the length of I.

Theorem. f is computable iff there is a computably enumerable set S of pairs (I_1, I_2) such that:

- Whenever $(I_1, I_2) \in S$, $f[I_1] \subseteq I_2$.
- For every $x \in \mathbb{R}$ and $\varepsilon > 0$, there is an $(I_1, I_2) \in S$ such that $I_1 \ni x$ and $|I_2| < \varepsilon$.

Proof. $\Rightarrow$: Look through Turing computations on names. $\Leftarrow$: use the information to compute $f(x)$.

Computable functions on $\mathbb{R}$

Theorem. A computable function from $\mathbb{R}$ to $\mathbb{R}$ is continuous.

Proof. Immediate from the second characterization.

Computable functions on $\mathbb{R}$

Most real numbers arising "in nature" are computable:

$$\pi, e, \gamma, \phi, \ldots$$

Similarly, continuous functions arising in nature are computable:

- polynomials
- trigonometric functions
- exp, log
- absolute value, min, and max
- solutions to ordinary differential equations with Lipschitz-continuous computable functions

Computable functions on $\mathbb{R}$

Note that the function

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

is not computable. In other words, we cannot decide $x \geq 0$.

Fix a small δ. We can do the next best thing, that is, decide

- $x \geq 0$
- $x \leq \delta$

Note that there is a "grey area" where either answer is o.k.

(Note also that the procedure cannot be extensional.)

Computability on computable reals

Let $\mathbb{R}_c$ denote the computable reals. One can also consider various notions of "computability" for functions $f : \mathbb{R}_c \to \mathbb{R}_c$.

For example:

- One can restrict the previous notion of computability to $\mathbb{R}_c$.
- One can consider $f : \mathbb{R}_c \to \mathbb{R}_c$ represented by a computable function φ_f that take an *index* e of a *computable* real, and return an index $\varphi_f(e)$ of a computable real.

There are functions that are computable (in our sense) when restricted to $\mathbb{R}_c$, but do not extend to a continuous function on $\mathbb{R}$.

Computability on computable reals

Say φ is *extensional* if whenever e_1 and e_2 represent the same real, so do $\varphi(e_1)$ and $\varphi(e_2)$.

Theorem (Kreiel, Lacomb, Schoenfield). If φ is extensional on all the computable reals, then φ is the restriction of a computable function on all of $\mathbb{R}$.

So any computable function in the second sense is computable in the first.

There is an extension due Ceĭtin to functions with a "computable separable domain."

See the tutorial on computable analysis by Brattka, Hertling, and Weihrauch.

Computable analysis

An important fact:

Theorem. Every computable function f on a compact interval $[a, b]$ has a computable *modulus of (uniform) continuity*, $M(\varepsilon)$:

$$\forall x, y \in [a, b] \, \forall \varepsilon > 0 \; |x - y| < M(\varepsilon) \rightarrow |f(x) - f(y)| < \varepsilon.$$

In fact, one can present a computable function by giving rational approximations on dyadic reals and a modulus of continuity.

The proof uses the Heine-Borel theorem: any cover of $[a, b]$ by open intervals has a finite subcover.

Compactness

Consider the contrapositive of the Heine-Borel theorem: if $(u_1, v_1), (u_2, v_2), \ldots$ does not cover $[0, 1]$, then there is a point x not in any of them.

This is *not* computationally true: there are computable sequences (u_i, v_i) that do not cover $[0, 1]$ but contain every computable x in $[0, 1]$.

Compactness

To simplify, let's shift focus to Cantor space, 2^ω with the product topology (where $2 = \{0, 1\}$).

Elements of 2^ω are infinite binary sequences, like $0110100010\ldots$.

Basic open sets are of the form $[\sigma]$, where $\sigma \in 2^{<\omega}$ is a finite binary sequence, and

$$[\sigma] = \{x \in 2^\omega \mid x \text{ extends } \sigma\}.$$

A *tree on* $\{0, 1\}$ is a set of binary sequences closed downwards.

Compactness

Trees code closed sets:

- If T is a tree, the set of paths through T is closed.
- If C is closed, $\{\sigma \mid [\sigma] \cap C \neq \emptyset\}$ is a tree.

If $\sigma_0, \sigma_1, \sigma_2, \ldots$ is a computable sequence, the set of x that do not extend any σ is given by the set of paths through a computable tree on $0, 1$.

At level n, enumerate $\sigma_0, \ldots, \sigma_n$, and keep all nodes that have not been ruled out.

Compactness

Theorem (Kleene). There is a computable infinite binary tree with no computable path.

Hint: Let $S_0 = \{e \mid \varphi_e(0)\downarrow = 0\}$, and $S_1 = \{e \mid \varphi_e(0)\downarrow = 1\}$. Show

- There is no computable set separating S_0 from S_1, i.e. containing S_0 and disjoint from S_1.
- There is a computable tree on $0, 1$ such that any path represents such a separation.

Easy: there is a path computable from the Halting problem.

The Low Basis Theorem (Jockusch, Soare): every such tree has a *low* path P, i.e. $P' = 0'$.

Compactness

Consider the "forward" statement: if a binary tree has no path, it is finite.

This is essentially Brouwer's "fan theorem."

It is "morally equivalent" to saying that if a sequence of intervals (a_i, b_i) covers $[0, 1]$, there is a finite subcover.

Bishop avoided it: for him, a continuous function on $[0, 1]$ has a modulus of uniform continuity *by definition*.

(This is an instance of Bishop's "avoidance of pseudogenerality.")

Computability in analysis

What is special about $\mathbb{R}$?

- There is a countable dense subset, $\mathbb{Q}$.
- With a natural encoding of $\mathbb{Q}$, operations we care about, including the distance function, are computable.
- One can construct $\mathbb{R}$ as the Cauchy completion of $\mathbb{Q}$.

The idea generalizes to separable metric spaces, and structures built on these.

Complete separable metric spaces

A *complete separable metric space* X can be presented by a countable subset A together with a function $d : A \times A \to \mathbb{R}$ satisfying:

- $d(x, x) = 0$
- $d(x, y) = d(y, x)$
- $d(x, z) \leq d(x, y) + d(y, z)$.

$X = \widehat{A}$ is the *completion* of A.

The *points of* $\widehat{A}$ are represented by sequences $(a_n)_{n \in \mathbb{N}}$ of elements of A such that for every n and $m \geq n$ we have $d(a_n, a_m) < 2^{-n}$.

(A *polish space* is a topological space that arises in this way.)

Computable metric spaces

Definition. A *computable metric space* is a triple $\mathcal{X} = (X, d, A)$:

- (X, d) is a complete, separable space
- $A = (a_i)_{n \in \mathbb{N}}$ is a countable dense subset
- The distances $d(a_i, a_j)$ are uniformly computable from i and j

Examples:

- $\mathbb{R}$, $\mathbb{C}$, $\mathbb{Q}_p$
- Infinite products, e.g. Baire Space, Cantor Space
- $C(X)$, for X a compact computable metric space
- function spaces, $L_1(X), L_2(X), L_p(X)$
- l_p, c_0

Computable spaces

Note that whether an element of X is computable depends on $(a_i)_{i \in \mathbb{N}}$, i.e. on both the dense subset and the way it is enumerated.

Such a choice gives a *representation* of X in terms of countable sequences of natural numbers.

More general presentations of computable analysis ("Type II effectivity") are phrased in terms of such representations. (Again, see the tutorial by Brattka et al.)

For our purposes, it is enough to consider separable metric spaces.

Computable metric spaces

Fix a computable separable metric space.

The computable elements and computable functions are defined as before.

For example, if (X, d, A) is a computable metric space, a computable function f from X to X is given by an algorithm that takes names to names.

Similarly for computable functions f from X to $\mathbb{R}$, etc.

Computable structures

We can extend these notions to other structures.

For example, a (real) *Banach space* is a vector space over $\mathbb{R}$ which is complete with respect to a norm.

A norm satisfies:

- $\|ax\| = |a| \cdot \|x\|$ for $a \in \mathbb{R}$
- $\|x + y\| \leq \|x\| + \|y\|$
- if $\|x\| = 0\|$, then $x = 0$.

Notice that $d(x, y) = \|x - y\|$ is a metric.

A *computable Banach space* is given by a computable metric spaces together with computable operations for scalar multiplication, addition, and the norm.

Computable structures

Notice that is is enough to compute the operations on the dense subset of "simple" elements.

Our examples are all Banach spaces:

- $\mathbb{R}$, $\mathbb{C}$, $\mathbb{Q}_p$
- Baire Space, Cantor Space
- $C(X)$, for X a compact metric space
- $L_1(X), L_2(X), L_p(X)$
- l_p, c_0

Computable structures

A (real) *Hilbert space* is a (real) Banach space that arises from an inner product:

- $\langle x, y \rangle = \langle y, x \rangle$
- $\langle ax, y \rangle = a\langle x, y \rangle$, $\langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle$
- $\langle x, x \rangle \geq 0$, and $\langle x, x \rangle = 0$ iff $x = 0$.

These guarantee $\|x\| = \langle x, x \rangle^{1/2}$ is a norm.

In a computable Hilbert space, the inner product is computable.

Computable measure spaces

How can we handle (finite) measure spaces? Think of $[0,1]$ with Lebesgue measure or $\{0,1\}^\omega$ with coin-flipping measure.

There are various options:

- define a countable algebra of "simple" sets $\mathcal{C}$, and a σ-additive measure μ on those, and take a completion
- define $L_1(X)$ as above, and define the measure space in terms of that
- take a computable measure to be an element of the space of measures under the *Prokhorov metric*

The approaches are equivalent, in the sense that one can translate between representations in a computable way.

See "Computability of probability measures and Martin-Löf randomness over metric spaces" by Hoyrup and Rojas.

Computability in mathematics

The benefit to defining notions of computable elements, objects, and structures is that we can now consider ordinary mathematical theorems from a computational perspective.

We have really done more:

- identified infinitary objects with countable sets of natural numbers
- defined the object to be computable if the corresponding set is

So we can also talk about computability relative to these objects, e.g. computable relative to a space or function.

Convergence theorems

A number of theorems of analysis assert that, under some hypotheses, a certain sequence (a_n) converges.

Suppose (a_n) is Cauchy:

$$\forall \varepsilon > 0 \; \exists m \; \forall n, n' \geq m \; d(a_{n'}, a_n) < \varepsilon$$

A function $r(\varepsilon)$ satisfying

$$\forall n, n' \geq r(\varepsilon) \; d(a_{n'}, a_n) < \varepsilon$$

is called a *bound on the rate of convergence*.

If there is a computable bound on the rate of convergence of (a_n), then (a_n) has a computable limit.

Rates of convergence

Note that the converse need not hold.

For example, one can design a sequence of rationals (a_n) that converges to 0, but there is no computable bound on the rate of convergence.

Question: which theorems of analysis have rates of convergence that are computable relative to the data?

Ergodic theory

A *discrete dynamical system* consists of a structure, $\mathcal{X}$, and an map T from $\mathcal{X}$ to $\mathcal{X}$:

- Think of the underlying set of $\mathcal{X}$ as the set of states of system.
- If x is a state, Tx gives the state after one unit of time.

In ergodic theory, $\mathcal{X}$ is assumed to be a finite measure space $(X, \mathcal{B}, \mu)$:

- $\mathcal{B}$ is a σ-algebra (the "measurable subsets").
- μ is a σ-additive measure, with $\mu(X) = 1$.

T is assumed to be a *measure preserving transformation*,
i.e. $\mu(T^{-1}A) = \mu(A)$ for every $A \in \mathcal{B}$.

Ergodic theory

Call $(X, \mathcal{B}, \mu, T)$ a *measure preserving system*.

- These can model physical systems (e.g. Hamilton's equations preserve Lebesgue measure).
- They can model probabilistic processes.
- They have applications to number theory and combinatorics.

The metamathematics of ergodic theory

Ergodic theory emerged from seventeenth century dynamics and nineteenth century statistical mechanics.

Since Poincaré, the emphasis has been on characterizing structural properties of dynamical systems, especially with respect to long term behavior (stability, recurrence).

Today, the field uses structural, infinitary, and nonconstructive methods that are characteristic of modern mathematics.

These are often at odds with computational concerns.

The metamathematics of ergodic theory

Central questions:

- To what extent can the methods and objects of ergodic theory be given a direct computational interpretation?
- How can we locate the "constructive content" of the nonconstructive methods?

Computable analysis addresses the first question, proof theory and proof mining the second.

The ergodic theorems

Consider the orbit x, Tx, $T^2x, \ldots$, and let $f : \mathcal{X} \to \mathbb{R}$ be some measurement. Consider the averages

$$\frac{1}{n}(f(x) + f(Tx) + \ldots + f(T^{n-1}x)).$$

For each $n \geq 1$, define $A_n f$ to be the function $\frac{1}{n}\sum_{i<n} f \circ T^i$.

Theorem (Birkhoff). For every f in $L^1(\mathcal{X})$, $(A_n f)$ converges pointwise almost everywhere, and in the L^1 norm.

A system is *ergodic* if for every A, $T^{-1}(A) = A$ implies $\mu(A) = 0$ or $\mu(A) = 1$.

If $\mathcal{X}$ is *ergodic*, then $(A_n f)$ converges to the constant function $\int f \; d\mu$.

The ergodic theorems

Recall that $L^2(\mathcal{X})$ is the Hilbert space of square-integrable functions on $\mathcal{X}$ modulo a.e. equality, with inner product

$$\langle f, g \rangle = \int fg \; d\mu$$

Theorem (von Neumann). For every f in $L^2(\mathcal{X})$, $(A_n f)$ converges in the L^2 norm.

A measure-preserving transformation T gives rise to an isometry $\widehat{T}$ on $L^2(\mathcal{X})$,

$$\widehat{T} f = f \circ T.$$

Riesz showed that the von Neumann ergodic theorem holds, more generally, for any nonexpansive operator $\widehat{T}$ on a Hilbert space (i.e. $\widehat{T}$ satisfies $\| \widehat{T} f \| \leq \| f \|$ for every f in $\mathcal{H}$).

Bounding the rate of convergence

Can we compute a bound on the rate of convergence of $(A_n f)$ from the inital data (T and f)?

In other words: can we compute a function $r : \mathbb{Q} \to \mathbb{N}$ such that for every rational $\varepsilon > 0$,

$$\|A_n f - A_{n'} f\| < \varepsilon$$

whenever $n, n' \geq r(\varepsilon)$?

Krengel (et al.): convergence can be arbitrarily slow.

But computability is a different question.

Noncomputability

Bishop gave a heuristic argument that shows that the ergodic theorem is not computable.

Imagine a tank divided in two, with red liquid on one side and white liquid on the other.

Imagine there is a (potential) leak, of size $\varepsilon \geq 0$.

If $\varepsilon > 0$, in the limit, the liquid is pink.

Predicting the limit amounts to testing whether $\varepsilon = 0$.

Noncomputability

It is not hard to turn this into a theorem. Here is one way:

Theorem (V'yugin). There is a computable shift-invariant measure μ on 2^ω such that there is no computable bound on the rate of convergence of $A_n 1_{[1]}$.

Here, $(2^\omega, T)$ is a familiar space, and we cook up a weird measure.

We can also consider a familiar measure, $(\mathbb{R}, \mu)$, and cook up a weird transformation.

Noncomputability

Theorem (Avigad and Simic). There are a computable measure-preserving transformation of $[0, 1]$ under Lebesgue measure and a computable characteristic function $f = \chi_A$, such that if $f^* = \lim_n A_n f$, then $\|f^*\|_2$ is not a computable real number.

In particular, f^* is not a computable element of $L^2(\mathcal{X})$, and there is no computable bound on the rate of convergence of $(A_n f)$ in either the L^2 or L^1 norm.

In general, everything is computable from $0'$, and this is sharp.

A positive result

Theorem (Avigad, Gerhardy, and Towsner). Let $\widehat{T}$ be a nonexpansive operator on a separable Hilbert space and let f be an element of that space. Let $f^* = \lim_n A_n f$. Then f^*, and a bound on the rate of convergence of $(A_n f)$ in the Hilbert space norm, can be computed from f, $\widehat{T}$, and $\|f^*\|$.

In particular, if $\widehat{T}$ arises from an ergodic transformation T, then f^* is computable from T and f.

Later on, we will “mine” additional positive information.

A positive result

Jason Rute and I showed that the previous result generalizes to uniformly convex Banach spaces (a much wider class).

The generalization builds on results of Kohlenbach and Leuștean, which in turn draws on a proof of the ergodic theorem by George Birkhoff.

Ideas:

- $\|f^*\|$ is equal to the infimum of $\|A_n f\|$.
- The norms $\|A_n f\|$ may oscillate, but if $\|A_n f\| = r$, then for every $\varepsilon > 0$, eventually $\|A_m f\| < r + \varepsilon$, and one can bound how long one has to wait.

So to compute a rate of convergence, wait until $\|A_n f\|$ is close to the limit, and then wait a little longer.

Classes of computable functions

Now, an interlude.

So far, we have been considering the general notion of computability.

We will also want to consider particular classes of computable functions.

Classes of computable functions

According to the Church-Turing thesis, Turing computability exhausts the intuitive notion of computability.

We have to allow *partial* functions, and we can't distinguish the total ones.

There are interesting subclasses of computable functions. We will consider two:

- The primitive recursive functions.
- The primitive recursive *functionals*.

Primitive recursive functions

The set of *primitive recursive functions* is the smallest set

- containing 0, $S(x) = x + 1$, $p_i^n(x_1, \ldots, x_n) = x_i$
- closed under composition
- closed under primitive recursion:

$$f(0, \vec{z}) = g(\vec{z}), \quad f(x + 1, \vec{z}) = h(f(x, \vec{z}), x, \vec{z})$$

Can define pairing and sequencing, and then handle operations on integers, rational numbers, lists, graphs, trees, finite sets, etc.

Anything “reasonably” computable (e.g. computable by a Turing machine in iterated exponential time) is primitive recursive.

Higher types

The *finite types* are defined as follows:

- N is a finite type
- If σ and τ are finite types, so are $\sigma \times \tau$ and $\sigma \to \tau$

For example, the reals can be represented as type $\mathsf{N} \to \mathsf{N}$ functionals. Functions from $\mathbb{R}$ to $\mathbb{R}$ can be represented by type $(\mathsf{N} \to \mathsf{N}) \to (\mathsf{N} \to \mathsf{N})$.

Higher types

Some useful notation:

- N is “type 0”
- N $\to$ N is “type 1”
- (N $\to$ N) $\to$ N is “type 2”
- $\ldots$

An object of type $n+1$ is, essentially (with currying and pairing), a functional $F(x_1, \ldots, x_k)$, taking arguments of type n, and returning a natural number.

This corresponds to $V_{\omega+n}$ in the set-theoretic hierarchy.

Higher types

Interpretations:

- The obvious set-theoretic interpretation:
- Hereditarily recursive objects (*HRO*)
  - $HRO_{\mathsf{N}} = \mathbb{N}$
  - $HRO_{\sigma \to \tau} = \{e \mid \forall x \ \in HRO_\sigma, \varphi_e(x) \in HRO_\tau\}$.
- Hereditarily extensional effective objects (*HEO*)
  Like *HRO*, but operations are required to respect (hereditary) extensional equality.

Higher type primitive recursion

The *primitive recursive functionals of finite type* allow:

- λ abstraction, application, pairing, projection
- Higher-type primitive recursion:

$$F(0) = G, \quad F(n+1) = H(F(n), n)$$

This allows us to talk about "primitive recursive real numbers" and "primitive recursive functions from $\mathbb{R}$ to $\mathbb{R}$.

Higher-type recursion

Ackermann's functions can be defined using primitive recursive functionals:

$$\begin{aligned}
F(0) &= S \\
F(n+1) &= \mathit{Iterate}(n+2, F(n)) \\
\mathit{Iterate}(0, G) &= \lambda x\, x \\
\mathit{Iterate}(n+1, G) &= \lambda x\, (G(\mathit{Iterate}(n, G)(x)))
\end{aligned}$$

Let $H(n) = F(n)(n)$.

Then H grows faster than any primitive recursive function.

Higher-type recursion

Once can restrict higher-type primitive recursion:

$$F(0, \vec{z}) = G(\vec{z}), \quad F(n+1, \vec{z}) = H(n, F(n, \vec{z}), \vec{z})$$

where $F(n, \vec{z})$ has type N.

The restricted primitive recursive functionals of type 1 are *exactly* the primitive recursive functions.