Proofs with Feasible Computational Content

Helmut Schwichtenberg

Mathematisches Institut der Universität München

Summer School Marktoberdorf
1. - 11. August 2007

Why extract computational content from proofs?

- Proofs are machine checkable $\Rightarrow$ no logical errors.
- Program on the proof level $\Rightarrow$ maintenance becomes easier. Possibility of program development by proof transformation (Goad 1980).
- Discover unexpected content:
  - Berger 1993: Tait's proof of the existence of normal forms for the typed λ-calculus $\Rightarrow$ "normalization by evaluation".
  - Content in weak (or "classical") existence proofs, of

    $$\tilde{\exists}_x A := \neg \forall_x \neg A,$$

    via proof interpretations: (refined) A-translation or Gödel's Dialectica interpretation.

Proof and computation

- $\to$, $\forall$, decidable prime formulas: negative arithmetic A^ω.
- Computational content (Brouwer, Heyting, Kolmogorov): by inductively defined predicates only. Examples: $\exists_x A$, $\mathrm{Acc}_\prec$.
- Induction $\sim$ (structural) recursion.
- Curry-Howard correspondence: formula $\sim$ type.
- Higher types necessary (nested $\to$, $\forall$).

Base types

$$
\begin{aligned}
\mathbf{U} &:= \mu_\alpha \alpha, \\
\mathbf{B} &:= \mu_\alpha(\alpha, \alpha), \\
\mathbf{N} &:= \mu_\alpha(\alpha, \alpha \to \alpha), \\
\mathbf{L}(\rho) &:= \mu_\alpha(\alpha, \rho \to \alpha \to \alpha), \\
\rho \wedge \sigma &:= \mu_\alpha(\rho \to \sigma \to \alpha), \\
\rho + \sigma &:= \mu_\alpha(\rho \to \alpha, \sigma \to \alpha), \\
(\mathrm{tree}, \mathrm{tlist}) &:= \mu_{\alpha,\beta}(\mathbf{N} \to \alpha, \beta, \beta \to \alpha, \alpha \to \beta \to \beta), \\
\mathrm{bin} &:= \mu_\alpha(\alpha, \alpha \to \alpha \to \alpha), \\
\mathcal{O} &:= \mu_\alpha(\alpha, \alpha \to \alpha, (\mathbf{N} \to \alpha) \to \alpha), \\
\mathcal{T}_0 &:= \mathbf{N}, \\
\mathcal{T}_{n+1} &:= \mu_\alpha(\alpha, (\mathcal{T}_n \to \alpha) \to \alpha).
\end{aligned}
$$

Types

Definition

$$\rho, \sigma, \tau ::= \mu \mid \rho \to \sigma.$$

A type is finitary if it is a base type

- with all its "parameter types" finitary, and
- all its "constructor types" without "functional" recursive argument types.

In the examples above $\mathbf{U}$, $\mathbf{B}$, $\mathbf{N}$, tree, tlist and bin are all finitary, but $\mathcal{O}$ and $\mathcal{T}_{n+1}$ are not. $\mathbf{L}(\rho)$ and $\rho \wedge \sigma$ are finitary if their parameter types ρ, σ are.

Recursion operators

$$
\begin{aligned}
&\mathtt{tt}^{\mathbf{B}} := \mathrm{C}_1^{\mathbf{B}}, \quad \mathtt{ff}^{\mathbf{B}} := \mathrm{C}_2^{\mathbf{B}},\\
&\mathcal{R}_{\mathbf{B}}^{\tau} \colon \mathbf{B} \to \tau \to \tau \to \tau,\\[1ex]
&0^{\mathbf{N}} := \mathrm{C}_1^{\mathbf{N}}, \quad \mathrm{S}^{\mathbf{N}\to\mathbf{N}} := \mathrm{C}_2^{\mathbf{N}},\\
&\mathcal{R}_{\mathbf{N}}^{\tau} \colon \mathbf{N} \to \tau \to (\mathbf{N} \to \tau \to \tau) \to \tau,\\[1ex]
&\mathrm{nil}^{\mathbf{L}(\rho)} := \mathrm{C}_1^{\mathbf{L}(\rho)}, \quad \mathrm{cons}^{\rho\to\mathbf{L}(\rho)\to\mathbf{L}(\rho)} := \mathrm{C}_2^{\mathbf{L}(\rho)},\\
&\mathcal{R}_{\mathbf{L}(\rho)}^{\tau} \colon \mathbf{L}(\rho) \to \tau \to (\rho \to \mathbf{L}(\rho) \to \tau \to \tau) \to \tau,\\[1ex]
&(\wedge_{\rho\sigma}^{+})^{\rho\to\sigma\to\rho\wedge\sigma} := \mathrm{C}_1^{\rho\wedge\sigma},\\
&\mathcal{R}_{\rho\wedge\sigma}^{\tau} \colon \rho\wedge\sigma \to (\rho \to \sigma \to \tau) \to \tau.
\end{aligned}
$$

We write $x :: l$ for $\mathrm{cons}\, x\, l$, and $\langle y, z\rangle$ for $\wedge^{+} yz$.

Terms and formulas

We work with typed variables $x^\rho, y^\rho, \ldots$.

Definition (Terms)

$$r, s, t ::= x^\rho \mid C \mid (\lambda_{x^\rho} r^\sigma)^{\rho \to \sigma} \mid (r^{\rho \to \sigma} s^\rho)^\sigma.$$

Definition (Formulas)

$$A, B, C ::= \mathrm{atom}(r^{\mathbf{B}}) \mid A \to B \mid \forall_{x^\rho} A.$$

atom is a predicate constant lifting a boolean term into a formula. Hence $\mathrm{atom}(r^{\mathbf{B}})$ means "r is true".

Examples

Projections:

$$t0 := \mathcal{R}^{\rho}_{\rho\wedge\sigma} t^{\rho\wedge\sigma}(\lambda_{x^\rho,y^\sigma} x^\rho), \quad t1 := \mathcal{R}^{\rho}_{\rho\wedge\sigma} t^{\rho\wedge\sigma}(\lambda_{x^\rho,y^\sigma} y^\sigma).$$

The append-function $:+:$ for lists is defined recursively by

$$\begin{aligned} \mathrm{nil} :+: l_2 &:= l_2, \\ (x :: l_1) :+: l_2 &:= x :: (l_1 :+: l_2). \end{aligned}$$

It can be defined as the term

$$l_1 :+: l_2 := \mathcal{R}^{\mathbf{L}(\alpha)\to\mathbf{L}(\alpha)}_{\mathbf{L}(\alpha)} l_1 (\lambda_{l_2} l_2)(\lambda_{x,l_1,p,l_2}(x :: (pl_2)))l_2.$$

Using the append function $:+:$ we can define list reversal R by

$$\begin{aligned} R\ \mathrm{nil} &:= \mathrm{nil}, \\ R(x :: l) &:= (R\ l) :+: (x :: \mathrm{nil}). \end{aligned}$$

The corresponding term is

$$R\ l := \mathcal{R}^{\mathbf{L}(\alpha)}_{\mathbf{L}(\alpha)} l\ \mathrm{nil}(\lambda_{x,l,p}(p :+: (x :: \mathrm{nil}))).$$

Induction

$$\mathrm{Ind}_{p,A} \colon \forall_p \big(A(\mathrm{tt}) \to A(\mathrm{ff}) \to A(p^{\mathbf{B}})\big),$$
$$\mathrm{Ind}_{n,A} \colon \forall_m \big(A(0) \to \forall_n (A(n) \to A(\mathrm{S}n)) \to A(m^{\mathbf{N}})\big),$$
$$\mathrm{Ind}_{l,A} \colon \forall_l \big(A(\mathrm{nil}) \to \forall_{x,l'} (A(l') \to A(x :: l')) \to A(l^{\mathbf{L}(\rho)})\big).$$

We also require the truth axiom $\mathrm{Ax}_{\mathrm{tt}} \colon \mathrm{atom}(\mathrm{tt})$.

Natural deduction: assumptions, $\to$-rules

derivation	term
$u\colon A$	u^A
$$\dfrac{\begin{array}{c}[u\colon A]\\ \mid M\\ B\end{array}}{A \to B}\to^+ u$$	$(\lambda_{u^A} M^B)^{A\to B}$
$$\dfrac{\begin{array}{c}\mid M\\ A\to B\end{array} \qquad \begin{array}{c}\mid N\\ A\end{array}}{B}\to^-$$	$(M^{A\to B} N^A)^B$

Natural deduction: $\forall$-rules

derivation	term
$$\dfrac{\begin{array}{c} \mid M \\ A \end{array}}{\forall_x A}\,\forall^+ x \quad (\text{VarC})$$	$(\lambda_x M^A)^{\forall_x A}$ (VarC)
$$\dfrac{\begin{array}{c} \mid M \\ \forall_x A(x) \end{array} \qquad r}{A(r)}\,\forall^-$$	$(M^{\forall_x A(x)} r)^{A(r)}$

Negative arithmetic A^ω

$\to$, $\forall$, decidable prime formulas. No inductively defined predicates.

$$\begin{aligned} F &:= \mathrm{atom}(\mathrm{ff}), \\ \neg A &:= A \to F, \\ \tilde{\exists}_x A &:= \neg \forall_x \neg A. \end{aligned}$$

Lemma (Stability, or principle of indirect proof)

$\vdash \neg\neg A \to A$, *for every formula* A *in* A^ω.

Proof.

Induction on A. For the atomic case one needs boolean induction (i.e., case distinction). □

An alternative: falsity as a predicate variable $\perp$

In A^ω, we have an "arithmetical" falsity $F := \mathrm{atom}(\mathrm{ff})$. However, in some proofs no knowledge about F is required. Then a predicate variable $\perp$ instead of F will do, and we can define

$$\tilde{\exists}_x A := \forall_x (A \to \perp) \to \perp.$$

Why is this of interest? We then can substitute an arbitrary formula for $\perp$, for instance, $\exists_x A$ (the "proper" existential quantifier, to be defined below). Then a proof of $\tilde{\exists}_x A$ is turned into a proof of

$$\forall_x (A \to \exists_x A) \to \exists_x A.$$

The premise will be provable. Hence we have a proof of $\exists_x A$.

2. Realizability interpretation

- Study the "computational content" of a proof.
- This only makes sense after we have added inductively defined predicates to our "negative" $\to, \forall$-language of A^ω.
- The resulting system will be called arithmetic with inductively defined predicates, ID^ω.

Example of an inductively defined predicate

Consider the graph of the list reversal function. The clauses or introduction axioms are

$$\begin{aligned}
&\mathrm{Rev}_0^+ : \forall^{\mathsf{U}}_{v,w}(F \to \mathrm{Rev}(v,w)),\\
&\mathrm{Rev}_1^+ : \mathrm{Rev}(\mathrm{nil},\mathrm{nil}),\\
&\mathrm{Rev}_2^+ : \forall^{\mathsf{U}}_{v,w}\forall_x(\mathrm{Rev}(v,w) \to \mathrm{Rev}(v :+: x:, x :: w)).
\end{aligned}$$

The (strengthened) elimination axiom says that Rev is the least predicate satisfying the clauses:

$$\begin{aligned}
\mathrm{Rev}^- : \forall^{\mathsf{U}}_{v,w}\big(&\forall^{\mathsf{U}}_{v,w}(F \to P(v,w)) \to\\
&P(\mathrm{nil},\mathrm{nil}) \to\\
&\forall^{\mathsf{U}}_{v,w}\forall_x\big(\mathrm{Rev}(v,w) \to P(v,w) \to P(v :+: x:, x :: w)\big) \to\\
&\mathrm{Rev}(v,w) \to P(v,w)\big).
\end{aligned}$$

The intended meaning of an inductively defined predicate I

- The clauses correspond to constructors of an appropriate algebra μ (better: μ_I).
- We associate to I of arity $\vec{\rho}$ a new predicate $I^{\mathbf{r}}$, of arity $(\mu, \vec{\rho})$, where the first argument r of type μ represents a generation tree, witnessing how the other arguments $\vec{r}$ were put into I.
- This object r of type μ is called a realizer of the prime formula $I(\vec{r})$.

Example (continued)

Recall the clauses for the graph of the list reversal function

$$\begin{aligned}
&\mathrm{Rev}_0^+ : \forall^{\mathsf{U}}_{v,w}(F \to \mathrm{Rev}(v,w)),\\
&\mathrm{Rev}_1^+ : \mathrm{Rev}(\mathrm{nil},\mathrm{nil}),\\
&\mathrm{Rev}_2^+ : \forall^{\mathsf{U}}_{v,w}\forall_x(\mathrm{Rev}(v,w) \to \mathrm{Rev}(v :+: x:, x :: w)).
\end{aligned}$$

The algebra μ_{Rev} is generated by

- two constants for the first two clauses, and
- a constructor of type $\mathbf{N} \to \mu_{\mathrm{Rev}} \to \mu_{\mathrm{Rev}}$ for the final clause.

Uniformity

- We want to select relevant parts of the computational content of a proof.
- This will be possible if some uniformities hold; we express this fact by using a uniform variant $\forall^{\mathsf{U}}$ of $\forall$ (as done by Berger 2005) and $\to^{\mathsf{U}}$ of $\to$.
- Both are governed by the same rules as the non-uniform ones. However, we will put some uniformity conditions on a proof to ensure that the extracted computational content is correct.

Example: existential quantifier

The (proper) existential quantifier is introduced as an inductively defined predicate with parameters. We have four variants, whose introduction axioms are

$$
\begin{aligned}
\exists^+ &: \quad \forall_x (A \to \exists_x A), \\
(\exists^{\mathsf{L}})^+ &: \forall_x (A \to^{\mathsf{U}} \exists^{\mathsf{L}}_x A), \\
(\exists^{\mathsf{R}})^+ &: \forall^{\mathsf{U}}_x (A \to \exists^{\mathsf{R}}_x A), \\
(\exists^{\mathsf{U}})^+ &: \forall^{\mathsf{U}}_x (A \to^{\mathsf{U}} \exists^{\mathsf{U}}_x A).
\end{aligned}
$$

Here $\exists_x A$ abbreviates $\mathrm{Ex}(\rho, \{ x^\rho \mid A \})$ (similar for the others).

Example: existential quantifier (continued)

The elimination axioms are (with $x \notin \mathrm{FV}(C)$)

$$
\begin{aligned}
\exists^-: &\quad \exists_x A \to \forall_x (A \to C) \to C, \\
(\exists^{\mathsf{L}})^-: &\ \exists_x^{\mathsf{L}} A \to \forall_x (A \to^{\mathsf{U}} C) \to C, \\
(\exists^{\mathsf{R}})^-: &\ \exists_x^{\mathsf{R}} A \to \forall_x^{\mathsf{U}} (A \to C) \to C, \\
(\exists^{\mathsf{U}})^-: &\ \exists_x^{\mathsf{U}} A \to \forall_x^{\mathsf{U}} (A \to^{\mathsf{U}} C) \to C.
\end{aligned}
$$

Example: Leibniz equality

The introduction axioms are

$$\mathrm{Eq}_0^+ : \forall^{\mathsf{U}}_{n,m}(F \to \mathrm{Eq}(n,m)), \quad \mathrm{Eq}_1^+ : \forall^{\mathsf{U}}_{n}\mathrm{Eq}(n,n),$$

and the elimination axiom is

$$\mathrm{Eq}^- : \forall^{\mathsf{U}}_{n,m}\big(\mathrm{Eq}(n,m) \to \forall^{\mathsf{U}}_{n} P(n,n) \to P(n,m)\big).$$

One can prove symmetry, transitivity and compatibility of Eq:

Lemma (CompatEq)

$\forall^{\mathsf{U}}_{n,m}\big(\mathrm{Eq}(n,m) \to Q(n) \to Q(m)\big).$

Proof.

Use Eq^-, with $P(n,m) := Q(n) \to Q(m)$. $\square$

Example: falsity

This example is somewhat extreme: the only introduction axiom is

$$\perp^{+}_{\mathrm{id}} \colon F \longrightarrow \perp_{\mathrm{id}}$$

and the elimination axiom

$$\perp^{-}_{\mathrm{id}} \colon (F \to C) \to \perp_{\mathrm{id}} \to C.$$

Example: pointwise equality $=_\rho$

For every arrow type $\rho \to \sigma$ we have the introduction axiom

$$\forall^{\mathrm{U}}_{x_1,x_2}\big(\forall_y(x_1 y =_\sigma x_2 y) \to x_1 =_{\rho\to\sigma} x_2\big).$$

Introduction axioms for $=_\mu$: Example with $\mathbf{T} := \mathcal{T}_1$:

$$\begin{aligned}
&\forall^{\mathrm{U}}_{x_1,x_2}(F \to x_1 =_{\mathbf{T}} x_2),\\
&0 =_{\mathbf{T}} 0,\\
&\forall^{\mathrm{U}}_{f_1,f_2}(\forall_n(f_1 n =_{\mathbf{T}} f_2 n) \to \mathrm{Sup} f_1 =_{\mathbf{T}} \mathrm{Sup} f_2).
\end{aligned}$$

The elimination axiom is

$$\begin{aligned}
=^-_{\mathbf{T}} : \forall^{\mathrm{U}}_{x_1,x_2}\big(x_1 =_{\mathbf{T}} x_2 \to P(0,0) \to&\\
&\forall^{\mathrm{U}}_{f_1,f_2}\big(\forall_n(f_1 n =_{\mathbf{T}} f_2 n) \to \forall_n P(f_1 n, f_2 n) \to\\
&\qquad P(\mathrm{Sup} f_1, \mathrm{Sup} f_2)\big) \to\\
&P(x_1,x_2)\big).
\end{aligned}$$

Example: pointwise equality (continued)

One can prove reflexivity of $=_\rho$, using meta-induction on ρ:

Lemma (ReflPtEq)

$\forall_n (n =_\rho n)$.

A consequence is that Leibniz equality implies pointwise equality:

Lemma (EqToPtEq)

$\forall_{n_1,n_2} \bigl(\mathrm{Eq}(n_1, n_2) \to n_1 =_\rho n_2\bigr)$.

Proof.

Use CompatEq and ReflPtEq. $\square$

Axioms

We express extensionality of our intended model by stipulating that pointwise equality implies Leibniz equality:

$$\texttt{PtEqToEq}\colon \forall_{n_1,n_2}\bigl(n_1 =_\rho n_2 \to \mathrm{Eq}(n_1, n_2)\bigr).$$

This implies

Lemma (CompatPtEqFct)

$\forall_f \forall^{\mathsf{U}}_{n_1,n_2}(n_1 =_\rho n_2 \to fn_1 =_\sigma fn_2)$.

Proof.

We obtain $\mathrm{Eq}(n_1, n_2)$ by PtEqToEq. By ReflPtEq we have $fn_1 =_\sigma fn_1$, hence $fn_1 =_\sigma fn_2$ by CompatEq. □

We write $\mathrm{E\text{-}ID}^\omega$ when the extensionality axioms are present.

In $\mathrm{E\text{-}ID}^\omega$ we can prove properties of the constructors of base types: they are injective, and have disjoint ranges.

Axioms (continued)

Let $\breve{\exists}$ denote any of $\exists, \exists^{\mathsf{R}}, \exists^{\mathsf{L}}, \exists^{\mathsf{U}}$. When $\breve{\exists}$ appears more than once, it is understood that it denotes the same quantifier each time. The axiom of choice (AC) is the scheme

$$\forall_{x^\rho} \breve{\exists}_{y^\sigma} A(x, y) \to \breve{\exists}_{f^{\rho\to\sigma}} \forall_{x^\rho} A(x, f(x)).$$

Independence axioms express the intended meaning of uniformities. The independence of premise axiom (IP) is

$$(A \to^{\mathsf{U}} \breve{\exists}_x B) \to \breve{\exists}_x (A \to^{\mathsf{U}} B) \quad (x \notin \mathrm{FV}(A)).$$

Similarly we have an independence of quantifier axiom (IQ) axiom

$$\forall_x^{\mathsf{U}} \breve{\exists}_y A \to \breve{\exists}_y \forall_x^{\mathsf{U}} A.$$

3. Computational content

We define simultaneously

- the type $\tau(A)$ of a formula A;
- when a formula is computationally relevant;
- the formula z realizes A, written $z \mathbf{r} A$, for a variable z of type $\tau(A)$;
- when a formula is negative;
- when an inductively defined predicate requires witnesses;
- for an inductively defined I requiring witnesses, its base type μ_I;
- for an inductively defined predicate I of arity $\vec{\rho}$ requiring witnesses, a witnessing predicate $I^{\mathbf{r}}$ of arity $(\mu_I, \vec{\rho})$.

The type of a formula

- Every formula A possibly containing inductively defined predicates can be seen as a computational problem. We define $\tau(A)$ as the type of a potential realizer of A, i.e., the type of the term (or program) to be extracted from a proof of A.
- More precisely, we assign to A an object $\tau(A)$ (a type or the "nulltype" symbol ε). In case $\tau(A) = \varepsilon$ proofs of A have no computational content.

$$\tau(\mathrm{atom}(r)) := \varepsilon, \quad \tau(I(\vec{r})) := \begin{cases} \varepsilon & \text{if } I \text{ does not require witnesses} \\ \mu_I & \text{otherwise,} \end{cases}$$

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

$$\tau(A \to^{\mathrm{U}} B) := \tau(B), \quad \tau(\forall^{\mathrm{U}}_{x^\rho} A) := \tau(A)$$

with the convention

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

Realizability

Let A be a formula and z either a variable of type $\tau(A)$ if it is a type, or the nullterm symbol ε if $\tau(A) = \varepsilon$. We define the formula $z \mathbf{r} A$, to be read z realizes A. The definition uses $I^{\mathbf{r}}$.

$$
\begin{aligned}
z \mathbf{r} \operatorname{atom}(s) &:= \operatorname{atom}(s),\\
z \mathbf{r} I(\vec{s}\,) &:= \begin{cases} I(\vec{s}\,) & \text{if } I \text{ does not require witnesses}\\ I^{\mathbf{r}}(z, \vec{s}\,) & \text{if not,}\end{cases}\\
z \mathbf{r} (A \to B) &:= \forall_x (x \mathbf{r} A \ \to\ zx \mathbf{r} B),\\
z \mathbf{r} (\forall_x A) &:= \forall_x\, zx \mathbf{r} A,\\
z \mathbf{r} (A \to^{\mathsf{U}} B) &:= (A \to z \mathbf{r} B),\\
z \mathbf{r} (\forall_x^{\mathsf{U}} A) &:= \forall_x\, z \mathbf{r} A
\end{aligned}
$$

with the convention $\varepsilon x := \varepsilon$, $z\varepsilon := z$, $\varepsilon\varepsilon := \varepsilon$.
Formulas without inductively defined predicates requiring witnesses are called negative. Example: $z \mathbf{r} A$. For A negative, $(\varepsilon \mathbf{r} A) = A$.

Witnesses

Definition (Uniform one-clause inductive definition)

- there is at most one clause apart from an efq-clause, and
- this clause is uniform, i.e., contains no $\forall$ but $\forall^{\mathsf{U}}$ only, and its premises are either negative or followed by $\to^{\mathsf{U}}$.

Examples: $\exists^{\mathsf{U}}$, $\bot_{\mathrm{id}}$, Eq.

An inductively defined predicate requires witnesses if it is not one of those, and not one of the predicates $I^{\mathbf{r}}$ introduced below.

For an inductively defined predicate I requiring witnesses, we define μ_I to be the corresponding component of the types $\vec{\mu} = \mu_{\vec{\alpha}}\vec{\kappa}$ generated from "constructor types" $\kappa_i := \tau(K_i)$ for all "constructor formulas" $K_0, \dots K_{k-1}$ from $\vec{I} = \mu_{\vec{X}}(K_0, \dots K_{k-1})$.

Extracted terms and uniform derivations

We define the extracted term of a derivation, and (using this concept) the notion of a uniform proof, which gives a special treatment to the uniform universal quantifier $\forall^{\mathsf{U}}$ and uniform implication $\to^{\mathsf{U}}$.

More precisely, for a proof M in $\mathrm{ID}^{\omega} + \mathrm{AC} + \mathrm{IP} + \mathrm{IQ}$, we simultaneously define

- its extracted term $[\![M]\!]$, of type $\tau(A)$, and
- when M is uniform.

Extracted terms and uniform proofs

For derivations M^A where $\tau(A) = \varepsilon$ (i.e., A is a Harrop formula) let $[\![M]\!] := \varepsilon$ (the nullterm symbol); every such M is uniform. Now assume that M derives a formula A with $\tau(A) \neq \varepsilon$. Then

$$\begin{aligned}
[\![u^A]\!] &:= x_u^{\tau(A)} \quad (x_u^{\tau(A)} \text{ uniquely associated with } u^A),\\
[\![(\lambda_{u^A} M)^{A\to B}]\!] &:= \lambda_{x_u^{\tau(A)}}[\![M]\!],\\
[\![M^{A\to B}N]\!] &:= [\![M]\!][\![N]\!],\\
[\![(\lambda_{x^\rho} M)^{\forall_x A}]\!] &:= \lambda_{x^\rho}[\![M]\!],\\
[\![M^{\forall_x A}r]\!] &:= [\![M]\!]r,\\
[\![(\lambda^{\mathsf{U}}_{u^A} M)^{A\to^{\mathsf{U}} B}]\!] &:= [\![M^{A\to^{\mathsf{U}} B}N]\!] := [\![(\lambda^{\mathsf{U}}_{x^\rho} M)^{\forall^{\mathsf{U}}_x A}]\!] := [\![M^{\forall^{\mathsf{U}}_x A}r]\!] := [\![M]\!].
\end{aligned}$$

In all these cases uniformity is preserved, except possibly in those involving λ^{U}:

Extracted terms and uniform proofs (continued)

Consider

$$\dfrac{\begin{matrix}[u\colon A] \\ \mid M \\ B\end{matrix}}{A \to^{\mathsf{U}} B}\,(\to^{\mathsf{U}})^+\, u \qquad \text{or as term} \quad (\lambda^{\mathsf{U}}_{u^A} M)^{A \to^{\mathsf{U}} B}.$$

$(\lambda^{\mathsf{U}}_{u^A} M)^{A \to^{\mathsf{U}} B}$ is uniform if M is and $x_u \notin \mathrm{FV}([\![M]\!])$. Similarly: Consider

$$\dfrac{\begin{matrix}\mid M \\ A\end{matrix}}{\forall^{\mathsf{U}}_x A}\,(\forall^{\mathsf{U}})^+\, x \qquad \text{or as term} \quad (\lambda^{\mathsf{U}}_x M)^{\forall^{\mathsf{U}}_x A} \qquad \text{(VarC)}.$$

$(\lambda^{\mathsf{U}}_x M)^{\forall^{\mathsf{U}}_x A}$ is uniform if M is and $x \notin \mathrm{FV}([\![M]\!])$.

Extracted terms for axioms

The extracted term of an induction axiom is defined to be a recursion operator. For example, in case of an induction scheme

$$\mathrm{Ind}_{n,A} \colon \forall_m \big(A(0) \to \forall_n (A(n) \to A(\mathrm{S}n)) \to A(m^{\mathbf{N}})\big)$$

we have

$$[\![\mathrm{Ind}_{n,A}]\!] := \mathcal{R}^{\tau}_{\mathbf{N}} \colon \mathbf{N} \to \tau \to (\mathbf{N} \to \tau \to \tau) \to \tau \quad (\tau := \tau(A) \neq \varepsilon).$$

For the introduction/elimination axioms of an inductively defined predicate I we define

$$[\![(I_j)^+_i]\!] := \mathrm{C}, \quad [\![I^-_j]\!] := \mathcal{R}_j,$$

and similary for the introduction and elimination axioms for $I^{\mathbf{r}}$. As extracted terms of (AC), (IP) and (IQ) we take identities of the appropriate types.

Uniform derivations

Lemma

There are purely logical uniform derivations of

- $A \to B$ *from* $\mathrm{A} \to^{\mathsf{U}} B$;
- $A \to^{\mathsf{U}} B$ *from* $\mathrm{A} \to B$, *provided* $\tau(A) = \varepsilon$ *or* $\tau(B) = \varepsilon$;
- $\forall_x A$ *from* $\forall_x^{\mathsf{U}} A$;
- $\forall_x^{\mathsf{U}} A$ *from* $\forall_x A$, *provided* $\tau(A) = \varepsilon$.

In formulas involving $\to^{\mathsf{U}}$ and $\forall^{\mathsf{U}}$ we can replace a subformula by an equivalent one:

Lemma

There are purely logical uniform derivations of

- $(A \to^{\mathsf{U}} B) \to (B \to B') \to A \to^{\mathsf{U}} B'$;
- $(A' \to A) \to^{\mathsf{U}} (A \to^{\mathsf{U}} B) \to A' \to^{\mathsf{U}} B$;
- $\forall_x^{\mathsf{U}} A \to (A \to A') \to \forall_x^{\mathsf{U}} A'$.

Characterization

When a formula A and its modified realizability interpretation $\exists_x x \mathbf{r} A$ are equivalent?

Theorem (Characterization)

In $\mathrm{ID}^\omega + \mathrm{AC} + \mathrm{IP} + \mathrm{IQ}$ *we can derive*

$$A \leftrightarrow \exists_x x \mathbf{r} A.$$

Proof.

Induction on A. $\square$

Soundness

Every theorem in $\text{E-ID}^\omega + \mathrm{AC} + \mathrm{IP} + \mathrm{IQ} + \mathrm{Ax}_\varepsilon$ has a realizer. Here $(\mathrm{Ax}_\varepsilon)$ is an arbitrary set of Harrop formulas (i.e., $\tau(A) = \varepsilon$) viewed as axioms.

Theorem (Soundness)

We work in $\mathrm{ID}^\omega + \mathrm{AC} + \mathrm{IP} + \mathrm{IQ}$. Let M be a derivation of A from assumptions $u_i \colon C_i$ $(i < n)$. Then we can find a derivation $\sigma(M)$ of $[\![M]\!]\ \mathbf{r}\ A$ from assumptions $\bar{u}_i \colon x_{u_i}\ \mathbf{r}\ C_i$ for a non-uniform u_i (i.e., $x_{u_i} \in \mathrm{FV}([\![M]\!])$), and $\bar{u}_i \colon C_i$ for the other ones.

Proof.

Induction on A. $\square$

Example: list reversal, constructive proof

View Rev as a variable for a binary boolean-valued function. It is axiomatized by

```
RevNil:    Rev(Nil nat)(Nil nat)
RevCons:   all v,w,x(Rev v w -> Rev(v:+:x:)(x::w))
```

Every non-empty list can be written in the form $v :+: y:$.

```
; "ListInitLastNat"
(set-goal (pf "all u,x ex v,y (x::u)=v:+:y:"))
```

Proof of $\forall_v \exists_w \mathrm{Rev}(v, w)$, by induction on $\mathrm{lh}(v)$.

Step: since the list is non-empty, it can be written as $v :+: y:$. v has a smaller length. Hence the IH yields its reversal w. Take $y :: w$.

Example: list reversal, constructive proof (continued)

```
; "ListRevNatEx"
(set-goal
 (pf "allnc Rev(
      Rev(Nil nat)(Nil nat) ->
      all v,w,x(Rev v w -> Rev(v:+:x:)(x::w)) ->
      all n,v(n=Lh v -> ex w Rev v w))"))
```

Extracted term:

```
[x0]
 (Rec nat=>list nat=>list nat)x0([v2](Nil nat))
 ([x2,f3,v4]
   [if v4
     (Nil nat)
     ([x5,v6][let p7 (cListInitLastNat v6 x5)
               (right p7::f3 left p7)])])
```

Example: list reversal, constructive proof (continued)

More readable form: Recursion equations for
$g :=$ `cListInitLastNat`:

$$
\begin{aligned}
g(\mathrm{nil}, z) &= (\mathrm{nil}, z), \\
g(x :: u, z) &= \textbf{let } (v, y) = g(u, x) \textbf{ in } (z :: v, y).
\end{aligned}
$$

Recursion equations for $h :=$ `cListRevNatEx`:

$$
\begin{aligned}
h(0, u) &= \mathrm{nil}, \\
h(n+1, \mathrm{nil}) &= \mathrm{nil}, \\
h(n+1, x :: u) &= \textbf{let } (v, y) = g(u, x) \textbf{ in } y :: h(n, v).
\end{aligned}
$$

We have extracted a quadratic algorithm.

```
(animate "ListInitLastNat")
(animate "Id")
(pp (nt (mk-term-in-app-form
          net (pt "4") (pt "1::2::3::4:"))))
; 4::3::2::1:
```

Example: list reversal, classical proof

From the "false" assumption $\forall_w (\mathrm{Rev}(v_0, w) \to \bot)$ we show that all initial segments of v_0 are non-revertible, by list induction:

```
; "InitSegNonRevStepU"
(set-goal
 (pf "all Rev(
  all v,w,x(Rev v w -> Rev(v:+:x:)(x::w)) ->
  all v0,x,u(
   allnc v(v:+:u=v0 -> all w(Rev v w -> bot)) ->
   allnc v(v:+:(x::u)=v0 -> all w(Rev v w -> bot))))"))

(pp (proof-to-extracted-term
     (theorem-name-to-proof "InitSegNonRevStepU")))
; [Rev,v0,x,u,h992,w]h992(x::w)
```

Note: `h992(x::w)` does not involve `v`. Hence `allnc v` is correct.

Example: list reversal, classical proof (continued)

```
; "InitSegNonRevU"
(set-goal
 (pf "all Rev(
  all v,w,x(Rev v w -> Rev(v:+:x:)(x::w)) ->
  all v0(
   all w(Rev v0 w -> bot) ->
   all u allnc v(v:+:u=v0 -> all w(Rev v w -> bot))))"))

; "RevClassU"
(set-goal
 (pf "all Rev,v(
      Rev(Nil nat)(Nil nat) ->
      all v,w,x(Rev v w -> Rev(v:+:x:)(x::w)) ->
      excl w Rev v w)"))
```

Example: list reversal, classical proof (continued)

- Substitute $\exists_w \mathrm{Rev}(v, w)$ for $\bot$,
- insert the trivial proof of $\forall_w(\mathrm{Rev}(v, w) \to \exists_w \mathrm{Rev}(v, w))$,
- extract a term from the resulting proof of $\exists_w \mathrm{Rev}(v, w)$ and
- normalize it, after "animating" `InitSegNonRevU`, and `InitSegNonRevStepU`. Let `net` be the result.

```
(pp net)
; [Rev0,v1]
;  (Rec list nat=>list nat=>list nat)v1([v2]v2)
;    ([x2,v3,f4,v5]f4(x2::v5))
;  (Nil nat)
```

More readable form: $f(v_1) = g(v_1, \mathrm{nil})$ with

$$g(\mathrm{nil}, v_2) = v_2, \quad g(x :: v_1, v_2) = g(v_1, x :: v_2).$$

We have extracted the usual linear algorithm.

4. Complexity

- Practically far too high, already for ground type structural ("primitive") recursion.
- Bellantoni and Cook (1992) characterized the polynomial time functions by the primitive recursion scheme, separating the variables into two sorts, as proposed by Simmons (1988):
- Input (or normal) variables control the length of recursion.
- Output (or safe) variables mark positions where substitution is allowed.

Here: extension to higher types.

The fast growing hierarchy $\{F_\alpha\}_{\alpha<\varepsilon_0}$

Grzegorczyk 1953, Robbin 1965, Löb and Wainer 1970, S. 1971

$$F_\alpha(n) = \begin{cases} n+1 & \text{if } \alpha = 0 \\ F_{\alpha-1}^{n+1}(n) & \text{if } \mathrm{Succ}(\alpha) \\ F_{\alpha(n)}(n) & \text{if } \mathrm{Lim}(\alpha) \end{cases}$$

where $F_{\alpha-1}^{n+1}(n)$ is the $n+1$-times iterate of $F_{\alpha-1}$ on n.

- F_ω is the Ackermann function.
- F_{ε_0} grows faster than all functions definable in arithmetic.

The power of higher types: iteration functionals

Pure types ρ_n: defined by $\rho_0 := \mathbf{N}$ and $\rho_{n+1} := \rho_n \to \rho_n$.
Let x_n be of pure type ρ_n.

$$F_\alpha x_n \dots x_0 := \begin{cases} x_0 + 1 & \text{if } \alpha = 0 \text{ and } n = 0, \\ x_n^{x_0} x_{n-1} \dots x_0 & \text{if } \alpha = 0 \text{ and } n > 0, \\ F_{\alpha-1}^{x_0} x_n \dots x_0 & \text{if } \mathrm{Succ}(\alpha), \\ F_{\alpha(x_0)} x_n \dots x_0 & \text{if } \mathrm{Lim}(\alpha). \end{cases}$$

Lemma

$F_\alpha F_\beta = F_{\beta+\omega^\alpha}$. *Hence all F_α are definable from F_0's (= iterators).*

A two-sorted variant $\mathrm{T}(;)$ of Gödel's T

The two-sortedness restriction is lifted to higher types.

We shall work with two forms of arrow types and abstraction terms:

$$\begin{cases} \mathbf{N} \to \sigma \\ \lambda_n r \end{cases} \quad \text{as well as} \quad \begin{cases} \rho \multimap \sigma \\ \lambda_z r \end{cases}$$

and a corresponding syntactic distinction between input $n^{\mathbf{N}}$ and output $a^{\mathbf{N}}, z^{\rho}$ (typed) variables. Intuition:

- A function of type $\mathbf{N} \to \sigma$ may recurse on its argument, but
- a function of type $\mathbf{N} \multimap \sigma$ may not.

The types are

$$\rho, \sigma, \tau ::= \mathbf{N} \mid \mathbf{N} \to \rho \mid \rho \multimap \sigma.$$

The $\to$-free types are called safe.

Constants, terms

The constants are $0\colon \mathbf{N}$, $\mathrm{S}\colon \mathbf{N} \multimap \mathbf{N}$ and, for safe τ,

$$\mathcal{C}_\tau \colon \mathbf{N} \multimap \tau \multimap (\mathbf{N} \multimap \tau) \multimap \tau,$$
$$\mathcal{R}_\tau \colon \mathbf{N} \to \tau \multimap (\mathbf{N} \to \tau \multimap \tau) \multimap \tau.$$

The first argument of $\mathcal{R}$ is the input (recursion) argument. Hence $\mathbf{N} \to$.

$\mathrm{T}(;)$-terms (terms for short) are

$$r, s, t ::= x \mid C \mid (\lambda_n r)^{\mathbf{N} \to \sigma} \mid r^{\mathbf{N} \to \sigma} s^{\mathbf{N}} \ (s \text{ input term}) \mid$$
$$(\lambda_z r)^{\rho \multimap \sigma} \mid r^{\rho \multimap \sigma} s^{\rho}.$$

s is an input term if all its free variables are input variables.

Examples

Addition:

$$a + 0 := a, \quad a + (\mathrm{S}n) := \mathrm{S}(a + n).$$

Representing term:

$$t_{+} := \lambda_{a,n}.\mathcal{R}_{\mathbf{N}} na(\lambda_{n,p}.\mathrm{S}p) \colon \mathbf{N} \multimap \mathbf{N} \to \mathbf{N}.$$

Predecessor P:

$$t_{P} := \lambda_{a}.\mathcal{C}_{\mathbf{N}} a0(\lambda_{b} b) \colon \mathbf{N} \multimap \mathbf{N}.$$

Modified subtraction $\dot{-}$:

$$a \dot{-} 0 := a, \quad a \dot{-} (\mathrm{S}n) := P(a \dot{-} n).$$

Representing term:

$$t_{\dot{-}} := \lambda_{a,n}.\mathcal{R}_{\mathbf{N}} na(\lambda_{n,p}.Pp) \colon \mathbf{N} \to \mathbf{N}.$$

Example: bounded summation, exponential

Let $f(\vec{n}, n) := \sum_{i<n} g(\vec{n}, i)$, i.e.,

$$f(\vec{n}, 0) := 0, \quad f(\vec{n}, \mathrm{S}n) := f(\vec{n}, n) + g(\vec{n}, n).$$

Representing term:

$$t_f := \lambda_{\vec{n},n}.\mathcal{R}_{\mathbf{N}} n 0 (\lambda_{n,p}.p + (t_g \vec{n} n)) \colon \mathbf{N}^{(k+1)} \to \mathbf{N}$$

Let $B(n, a) := a + 2^n$, i.e.,

$$\begin{aligned} B(0, a) &= a + 1, \\ B(n+1, a) &= B(n, B(n, a)). \end{aligned}$$

Representing term:

$$t_B := \lambda_n.\mathcal{R}_{\mathbf{N} \multimap \mathbf{N}} n \mathrm{S} \big(\lambda_{m,p,a}(p^{\mathbf{N} \multimap \mathbf{N}}(pa))\big) \colon \mathbf{N} \to \mathbf{N} \multimap \mathbf{N}$$

Elementary functions are definable in $\mathrm{T}(;)$

The class $\mathcal{E}$ of elementary functions consists of those number theoretic functions which can be defined from

- the initial functions: constant 0, successor S, projections (onto the ith coordinate), addition $+$, modified subtraction $\dot{-}$, multiplication $\cdot$ and exponentiation 2^x
- by applications of composition and bounded minimization.

Bounded minimization

$$f(\vec{n}, m) = \mu_{k<m}(g(\vec{n}, k) = 0)$$

is definable from bounded summation and $\dot{-}$:

$$f(\vec{n}, m) = \sum_{i<m} \bigl(1 \dot{-} \sum_{k \leq i} (1 \dot{-} g(\vec{n}, k))\bigr).$$

The claim follows from the examples above.

Necessity of the restrictions on the type of $\mathcal{R}$

Define the pure safe types ρ_k, by $\rho_0 := \mathbf{N}$ and $\rho_{k+1} := \rho_k \multimap \rho_k$. In $\mathrm{T}(;)$ we can define

$$Ina_k \dots a_0 := a_k^n a_{k-1} \dots a_0,$$

with a_k of type ρ_k. However, a definition $F_0 a_k \dots a_0 := Ia_0 a_k \dots a_0$ is not possible: Ia_0 is not allowed.

The value type is a safe type:

$$I_E := \lambda_n.\mathcal{R}_{\mathbf{N}\to\mathbf{N}} n(\lambda_m m)\big(\lambda_{n,p,m}(p^{\mathbf{N}\to\mathbf{N}}(Em))\big),$$

and $I_E(n,m) = E^n(m)$, a function of superelementary growth.

The "previous"-variable is an output variable:

$$S := \lambda_n.\mathcal{R}_{\mathbf{N}} n0\big(\lambda_{n,m}(Em)\big)$$

Then $S(n) = E^n(0)$, which is superelementary.

Theorem (Normalization)

Let t be a closed $\mathrm{T}(;)$-term of type $\mathbf{N} \twoheadrightarrow \ldots \mathbf{N} \twoheadrightarrow \mathbf{N}$ *($\twoheadrightarrow \in \{\to, \multimap\}$). Then t denotes an elementary function.*

Proof.

- Let $\vec{x}$ be new variables such that $t\vec{x}$ is of type $\mathbf{N}$. The β normal form $\beta\text{-nf}(t\vec{x})$ of $t\vec{x}$ is computed in an amount of time that may be large, but it is only a constant with respect to $\vec{n}$.
- By $\mathcal{R}$ Elimination one reduces to an $\mathcal{R}$-free term $\mathrm{rf}(\beta\text{-nf}(t\vec{x}); \vec{x}; \vec{n})$ in time $F_t(|\vec{n}|)$ with F_t elementary.
- Since the running time bounds the size of the produced term, $|\mathrm{rf}(\beta\text{-nf}(t\vec{x}); \vec{x}; \vec{n})| \leq F_t(|\vec{n}|)$.
- A further β-normalization computes $\beta\mathcal{R}\text{-nf}(t\vec{n}) = \beta\text{-nf}(\mathrm{rf}(\beta\text{-nf}(t\vec{x}); \vec{x}; \vec{n}))$ in time elementary in $|\vec{n}|$.
- Finally in time linear in the result we can remove all occurrences of $\mathcal{C}$ and arrive at a numeral.

A linear two-sorted variant $\mathrm{LT}(;)$ of Gödel's T

Work with a binary representation of the natural numbers, with two successors $S_0(a) = 2a$ and $S_1(a) = 2a + 1$.

Recall: for $B(n, a) = a + 2^n$ we had the defining term

$$\lambda_n \big(\mathcal{R}_{\mathbf{N} \multimap \mathbf{N}} n \mathrm{S} \big(\lambda_{m,p,a}(p^{\mathbf{N} \multimap \mathbf{N}}(pa))\big)\big)$$

with the higher type variable p for the "previous" value appearing twice in the step term. Here:

- The term definition will now involve a linearity constraint.
- Change type of $\mathcal{R}$: its (higher type) step argument will be used many times, and hence we need a $\to$ after it.

Change names: input/output $\mapsto$ normal/safe variables.

Feasible computation with higher types: $\mathrm{LT}(;)$

We work with two forms of arrow types and abstraction terms:

$$\begin{cases} \rho \to \sigma \\ \lambda_{\bar{x}^\rho} r \end{cases} \quad \text{as well as} \quad \begin{cases} \rho \multimap \sigma \\ \lambda_{x^\rho} r \end{cases}$$

and a corresponding syntactic distinction between normal and safe (typed) variables, $\bar{x}$ and x. Intuition:

- A function of type $\rho \to \sigma$
  - may recurse on its argument (if of ground type), or
  - use it many times (if of higher type).
- A function of type $\rho \multimap \sigma$
  - may not recurse on its argument (if of ground type), or
  - can use it only once (if of higher type).

Types

The types are

$$\rho, \sigma, \tau ::= \mathbf{U} \mid \mathbf{B} \mid \mathbf{L}(\rho) \mid \rho \to \sigma \mid \rho \multimap \sigma \mid \rho \land \sigma,$$

and the level of a type is defined by

$$\begin{aligned} l(\mathbf{U}) &:= 0, \\ l(\mathbf{B}) &:= 0, \\ l(\mathbf{L}(\rho)) &:= l(\rho), \end{aligned} \qquad \begin{aligned} l(\rho \to \sigma) := l(\rho \multimap \sigma) &:= \max\{l(\sigma), 1 + l(\rho)\}, \\ l(\rho \land \sigma) &:= \max\{l(\rho), l(\sigma)\}. \end{aligned}$$

The $\to$-free types are called safe.

Constants

The constants are $\mathbf{u} \colon \mathbf{U}$, $\mathtt{tt}, \mathtt{ff} \colon \mathbf{B}$, $\mathrm{nil}_\rho \colon \mathbf{L}(\rho)$ and, for safe ρ, τ,

$$
\begin{aligned}
&::_\rho \colon \rho \multimap \mathbf{L}(\rho) \multimap \mathbf{L}(\rho),\\
&\mathrm{if}_\tau \colon \mathbf{B} \multimap \tau \multimap \tau \multimap \tau,\\
&\mathcal{C}^\rho_\tau \colon \mathbf{L}(\rho) \multimap \tau \multimap (\rho \multimap \mathbf{L}(\rho) \multimap \tau) \multimap \tau,\\
&\mathcal{R}^\rho_\tau \colon \mathbf{L}(\rho) \to \tau \multimap (\rho \to \mathbf{L}(\rho) \to \tau \multimap \tau) \to \tau \quad (\rho \text{ ground}),\\
&\wedge^+_{\rho\sigma} \colon \rho \multimap \sigma \multimap \rho \wedge \sigma,\\
&\wedge^-_{\rho\sigma\tau} \colon \rho \wedge \sigma \multimap (\rho \multimap \sigma \multimap \tau) \multimap \tau,
\end{aligned}
$$

Terms

$\mathrm{LT}(;)$-terms are built from these constants and typed variables $\bar{x}^\sigma$ (normal variables) and x^σ (safe variables) by introduction and elimination rules for the two type forms $\rho \to \sigma$ and $\rho \multimap \sigma$, i.e.,

$\bar{x}^\rho \mid x^\rho \mid C^\rho \quad$ (constant) $\mid$
$(\lambda_{\bar{x}^\rho} r^\sigma)^{\rho \to \sigma} \mid (r^{\rho \to \sigma} s^\rho)^\sigma \quad$ (s "normal") $\mid$
$(\lambda_{x^\rho} r^\sigma)^{\rho \multimap \sigma} \mid (r^{\rho \multimap \sigma} s^\rho)^\sigma \quad$ (higher type safe variables in r, s distinct),

where a term s is called normal if all its free variables are normal.

Examples

$x \oplus y$ concatenates $|x|$ bits onto y:

$$1 \oplus y = S_0 y,$$
$$(S_i x) \oplus y = S_0(x \oplus y).$$

The representing term is

$$\bar{x} \oplus y := \lambda_{\bar{x},y}.\mathcal{R}_{\mathbf{W} \multimap \mathbf{W}} \bar{x} S_0(\lambda_{\bar{z},\bar{l},p,y}.S_0(p^{\mathbf{W} \multimap \mathbf{W}} y))y : \mathbf{W} \to \mathbf{W} \multimap \mathbf{W}.$$

$x \odot y$ has output length $|x| \cdot |y|$:

$$x \odot 1 = x,$$
$$x \odot (S_i y) = x \oplus (x \odot y).$$

The representing term is

$$\bar{x} \odot \bar{y} := \lambda_{\bar{x},\bar{y}}.\mathcal{R}_{\mathbf{W}} \bar{y} \bar{x} (\lambda_{\bar{z},\bar{l},p}.\bar{x} \oplus p) : \mathbf{W} \to \mathbf{W} \to \mathbf{W}.$$

Polytime computable functions are definable in $\mathrm{LT}(;)$

Bellantoni/Cook (1992) characterized the polynomial time computable functions: some initial functions, safe composition

$$f(\vec{x};\vec{y}) := g(r_1(\vec{x};),\dots,r_m(\vec{x};);s_1(\vec{x};\vec{y}),\dots,s_n(\vec{x};\vec{y}))$$

and safe recursion:

$$\begin{aligned} f(1,\vec{x};\vec{y}) &:= g(\vec{x};\vec{y}),\\ f(S_i n,\vec{x};\vec{y}) &:= h_i(n,\vec{x};\vec{y},f(n,\vec{x};\vec{y})). \end{aligned}$$

Representing term:

$$\begin{aligned} &t_f := \lambda_{\bar{n},\vec{\bar{x}}}.\mathcal{R}_\tau \bar{n}(t_g\vec{\bar{x}})s \quad \text{with}\\ &s := \lambda_{\bar{x},\bar{l},p,\vec{y}}.\mathrm{if}_{\mathbf{W}\multimap\mathbf{W}}\bar{x}(\lambda_z.t_{h_0}\bar{l}\vec{\bar{x}}\vec{y}z)(\lambda_z.t_{h_1}\bar{l}\vec{\bar{x}}\vec{y}z)(p\vec{y}). \end{aligned}$$

Note p is used only once.

Theorem (Normalization)

Let r be a closed $\mathrm{LT}(;)$-term of type $\mathbf{W} \twoheadrightarrow \dots \mathbf{W} \twoheadrightarrow \mathbf{W}$ *($\twoheadrightarrow \in \{\to, \multimap\}$). Then r denotes a polytime function.*

Proof.

- Let $\vec{x}$ be new variables of types $\vec{\rho}$. The normal form of $t\vec{x}$ is computed in an amount of time that may be large, but it is still only a constant with respect to $\vec{n}$.
- $\mathrm{nf}(t\vec{x})$ is "simple" (i.e., no higher type normal variables).
- By $\mathcal{R}$ Elimination one reduces to an $\mathcal{R}$-free simple term $\mathrm{rf}(\mathrm{nf}(t\vec{x}); \vec{x}; \vec{n})$ in time $P_t(|\vec{n}|)$, w.r.t. to a dag model of computation.
- Since the running time bounds the size of the produced term, $|\mathrm{rf}(\mathrm{nf}(t\vec{x}); \vec{x}; \vec{n})| \leq P_t(|\vec{n}|)$.
- By Sharing Normalization one computes $\mathrm{nf}(t\vec{n}) = \mathrm{nf}(\mathrm{rf}(\mathrm{nf}(t\vec{x}); \vec{x}; \vec{n}))$ in time $O(P_t(|\vec{n}|)^2)$.

$\square$

Future work

Arithmetic with inductively defined predicates: ID^{ω}.

- Fine tuning of computational content: $\forall^{\mathsf{U}}$ and $\to^{\mathsf{U}}$.
- Compare different proof interpretations: "refined" A-translation and Gödel's Dialectica interpretation.
- Solve
$$\frac{\text{Arithmetic}}{\text{Gödel's } \mathrm{T}} = \frac{\mathrm{A}(;)}{\mathrm{T}(;)} = \frac{\mathrm{LA}(;)}{\mathrm{LT}(;)}.$$
- Terms: Gödel's T over (possibly infinitary) base types, with structural and general recursion.
- Standard semantics: Partial continuous functionals. Terms denote computable functionals. Include formal neighborhoods (consistent sets in the sense of Scott's information systems) into the language.