Proofs with Feasible Computational Content

Helmut Schwichtenberg

Mathematisches Institut der Universität München

Summer School Marktoberdorf
1. - 11. August 2007

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 $\to$, $\forall$).

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 proofs of $\tilde{\exists}_x A$, via proof interpretations: (refined) A-translation or Gödel's Dialectica interpretation.

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

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

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 **U**, **B**, **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

$$\mathrm{tt}^{\mathbf{B}} := \mathrm{C}_1^{\mathbf{B}}, \quad \mathrm{ff}^{\mathbf{B}} := \mathrm{C}_2^{\mathbf{B}},$$
$$\mathcal{R}_{\mathbf{B}}^{\tau} \colon \mathbf{B} \to \tau \to \tau \to \tau,$$

$$0^{\mathbf{N}} := \mathrm{C}_1^{\mathbf{N}}, \quad \mathrm{S}^{\mathbf{N}\to\mathbf{N}} := \mathrm{C}_2^{\mathbf{N}},$$
$$\mathcal{R}_{\mathbf{N}}^{\tau} \colon \mathbf{N} \to \tau \to (\mathbf{N} \to \tau \to \tau) \to \tau,$$

$$\mathrm{nil}^{\mathbf{L}(\rho)} := \mathrm{C}_1^{\mathbf{L}(\rho)}, \quad \mathrm{cons}^{\rho\to\mathbf{L}(\rho)\to\mathbf{L}(\rho)} := \mathrm{C}_2^{\mathbf{L}(\rho)},$$
$$\mathcal{R}_{\mathbf{L}(\rho)}^{\tau} \colon \mathbf{L}(\rho) \to \tau \to (\rho \to \mathbf{L}(\rho) \to \tau \to \tau) \to \tau,$$

$$\left(\wedge^{+}_{\rho\sigma}\right)^{\rho\to\sigma\to\rho\wedge\sigma} := \mathrm{C}_1^{\rho\wedge\sigma},$$
$$\mathcal{R}_{\rho\wedge\sigma}^{\tau} \colon \rho \wedge \sigma \to (\rho \to \sigma \to \tau) \to \tau.$$

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)

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

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

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

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

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

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

Natural deduction: assumptions, $\to$-rules

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

Natural deduction: $\forall$-rules

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

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). $\square$

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

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 $\bot$ instead of F will do, and we can define

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

Why is this of interest? We then can substitute an arbitrary formula for $\bot$, for instance, $\exists_x A$ (the "proper" existential quantifier, to be defined below). Then

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

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

Realizability interpretation

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

The intended meaning of an inductively defined predicate I

- The clauses correspond to constructors of an appropriate algebra μ (or better μ_I).
- We associate to I 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

Consider the graph of the list reversal function as an inductively defined predicate. The clauses or introduction axioms are

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

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.

Example (continued)

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

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

Uniformity

- We want to select relevant parts of the complete 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 $\rightarrow^{\mathsf{U}}$ of $\rightarrow$.
- 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

Let α be a type variable, y an object variable of type α, and Q a predicate variable of arity (α). We have four variants:

$$\begin{aligned}
\mathrm{Ex}(\alpha, Q) &:= \mu X\big(\forall_y(Q(y) \to X)\big),\\
\mathrm{ExL}(\alpha, Q) &:= \mu X\big(\forall_y(Q(y) \to^{\mathsf{U}} X)\big),\\
\mathrm{ExR}(\alpha, Q) &:= \mu X\big(\forall^{\mathsf{U}}_y(Q(y) \to X)\big),\\
\mathrm{ExU}(\alpha, Q) &:= \mu X\big(\forall^{\mathsf{U}}_y(Q(y) \to^{\mathsf{U}} X)\big).
\end{aligned}$$

The introduction axioms are

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

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

Example: existential quantifier (continued)

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

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

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

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

One can prove symmetry, transitivity and compatibility of Eq:

Lemma (CompatEq)

$\forall^{\mathsf{U}}_{n_1,n_2}\big(\mathrm{Eq}(n_1,n_2) \to Q(n_1) \to Q(n_2)\big)$.

Proof.

Use Eq^-. □

Example: pointwise equality $=_\rho$

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

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

An example of $=_\mu$ with a non-finitary base type μ is $=_{\mathbf{T}}$ for $\mathbf{T} := \mathcal{T}_1$:

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

The elimination axiom is

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

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)

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

Proof.

Use CompatEq and ReflPtEq. □

Further axioms and their consequences

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

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

This implies

Lemma (CompatPtEqFct)

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

Proof.

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

We write $\mathrm{E\text{-}ID}^\omega$ when the extensionality axioms PtEqToEq are present. In $\mathrm{E\text{-}ID}^\omega$ we can prove properties of the constructors of our free algebras: that they are injective, and have disjoint ranges.

Further axioms

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

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

The 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

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

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 **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.

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

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

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

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 $I^{\mathbf{r}}$.

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

with the convention $\varepsilon x := \varepsilon$, $z\varepsilon := z$, $\varepsilon\varepsilon := \varepsilon$.
Formulas which do not contain inductively defined predicates requiring witnesses play a special role; we call them negative. Their crucial property is $(\varepsilon\ \mathbf{r}\ A) = A$. Every formula $z\ \mathbf{r}\ A$ is negative.

Witnesses

Consider a particularly simple inductively defined predicate, where

- 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 are $\exists^{\mathsf{U}}$, $\bot$, Eq. We call those predicates uniform one-clause defined. 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 uniform implication $\to^{\mathsf{U}}$ and the uniform universal quantifier $\forall^{\mathsf{U}}$. More precisely, for a derivation M in $\mathrm{ID}^\omega + \mathrm{AC} + \mathrm{IP}_\varepsilon + \mathrm{Ax}_\varepsilon$, we simultaneously define

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

Extracted terms

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

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

In all these cases uniformity is preserved, except possibly in those involving λ^{U}: $\lambda^{\mathrm{U}}_{u^A} M$ is uniform if M is and $x_u \notin \mathrm{FV}([\![M]\!])$, and $\lambda^{\mathrm{U}}_{x^\rho} M$ is uniform if M is and – in addition to the usual variable condition – $x \notin \mathrm{FV}([\![M]\!])$.

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* $\tau(A) = \varepsilon$.

We certainly want to know that 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

We consider the question when a formula A and its modified realizability interpretation $\exists_x\, x\ \mathbf{r}\ A$ are equivalent.

Theorem (Characterization)

$$\mathrm{ID}^\omega + \mathrm{AC} + \mathrm{IP} + \mathrm{IQ} \vdash A \leftrightarrow \exists_x\, x\ \mathbf{r}\ A.$$

Proof.

Induction on A. □

Soundness

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

Theorem (Soundness)

*Let M be a derivation of A from assumptions $u_i : C_i$ $(i < n)$. Then we can find a derivation $\sigma(M)$ of $[\![M]\!]$ **r** A from assumptions $\bar{u}_i : x_{u_i}$ **r** C_i for a non-uniform u_i (i.e., $x_{u_i} \in \mathrm{FV}([\![M]\!])$), and $\bar{u}_i : C_i$ for the other ones.*

Proof.

Induction on A. □

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 kinds.
- 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

- Goal: The functions definable in $\mathrm{T}(;)$ are exactly the elementary functions.
- Proof idea: β-normalization of terms of rank $\leq k$ has elementary complexity, and that the two-sortedness restriction allows to unfold $\mathcal{R}$ in a controlled way.

The approach of Simmons (1988) and Bellantoni/Cook (1992) is lifted to higher types.

Higher order terms with input/output restrictions

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 and output (typed) variables.. The types are

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

and the level of a type is defined by

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

Ground types are the types of level 0, and a higher type is any type of level at least 1. The $\to$-free types are called safe. In particular, every ground type is safe.

Constants

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

$$\begin{aligned}
&\mathrm{S} \colon \mathbf{N} \multimap \mathbf{N},\\
&\mathcal{C}_\tau \colon \mathbf{N} \multimap \tau \multimap (\mathbf{N} \multimap \tau) \multimap \tau,\\
&\mathcal{R}_\tau \colon \mathbf{N} \to \tau \multimap (\mathbf{N} \to \tau \multimap \tau) \multimap \tau.
\end{aligned}$$

Comments to the typing of $\mathcal{R}_\tau$:

- The first argument is the one that is recursed on and hence must be an input argument, so the type starts with $\mathbf{N} \to \cdots$.
- The third argument is the step argument; here we have used the type $\mathbf{N} \to \tau \multimap \tau$ rather than $\mathbf{N} \multimap \tau \multimap \tau$, because then we can construct a step term in the form $\lambda_{n,p} t$ rather than $\lambda_{a,p} t$, which is more flexible.

Variables and terms

We shall work with typed variables. A variable of type **N** is either an input or an output variable, and variables of a type different from **N** are always output variables. Conventions:

x	(input or output) variable;
z	output variable;
n, m	input variable of type **N**;
a	output variable of type **N**.

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

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

We call s an input term if all its free variables are input variables. C is a constant.

Conversions

$$
\begin{aligned}
\big(\lambda_{\vec{x},x} r(\vec{x}, x)\big)\vec{s}s &\mapsto \big(\lambda_{\vec{x}} r(\vec{x}, s)\big)\vec{s}, \\
\mathcal{C}_\tau 0ts &\mapsto t, \\
\mathcal{C}_\tau(\mathrm{S}r)ts &\mapsto sr, \\
\mathcal{R}_\tau 0ts &\mapsto t, \\
\mathcal{R}_\tau(\mathrm{S}r)ts &\mapsto sr(\mathcal{R}_\tau rts).
\end{aligned}
$$

Why not $(\lambda_x r(x))s \mapsto r(s)$? In some arguments (e.g., in the proof of the β-normalization theorem below) we need to perform conversions of highest level first, we must be able to convert $(\lambda_{\vec{x},x} r(\vec{x}, x))\vec{s}s$ with $\vec{x}$ of a low and x of a high level into $(\lambda_{\vec{x}} r(\vec{x}, s))\vec{s}$.

Normal forms, definable functions

- Redexes are subterms shown on the left side of the conversion rules above.
- We write $r \longrightarrow r'$ ($r \longrightarrow^* r'$) if r can be reduced into r' by one (an arbitrary number of) conversion of a subterm.
- A term is in normal form if it does not contain a redex as a subterm.

Definition

A function f is called definable in $\mathrm{T}(;)$ if there is a closed $\mathrm{T}(;)$-term $t_f : \mathbf{N} \twoheadrightarrow \ldots \mathbf{N} \twoheadrightarrow \mathbf{N}$ ($\twoheadrightarrow \in \{\longrightarrow, \multimap\}$) in $\mathrm{T}(;)$ denoting this function.

Notice that it is always desirable to have more $\multimap$ in the type of t_f, because then there are less restrictions on its argument terms.

Examples

Addition can be defined by a term t_+ of type $\mathbf{N} \multimap \mathbf{N} \to \mathbf{N}$. The recursion equations are

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

and the representing term is

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

The predecessor function P can be defined by a term t_P of type $\mathbf{N} \multimap \mathbf{N}$ if we use the cases operator $\mathcal{C}$:

$$t_P := \lambda_a.\mathcal{C}_{\mathbf{N}} a0(\lambda_b b).$$

From the predecessor function we can define modified subtraction $\dot{-}$:

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

by the term

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

Example: bounded summation

If f is defined from g by bounded summation
$f(\vec{n}, n) := \sum_{i<n} g(\vec{n}, i)$, i.e.,

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

and we have a term t_g of type $\mathbf{N}^{(k+1)} \longrightarrow \mathbf{N}$ defining g, then we can build a term t_f of type $\mathbf{N}^{(k+1)} \longrightarrow \mathbf{N}$ defining f by

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

Example: exponential growth

We now show that in spite of our restrictions on the formation of types and terms we can define functions of exponential growth. Probably the easiest function of exponential growth is $B(n, a) = a + 2^n$ of type $B\colon \mathbf{N} \to \mathbf{N} \multimap \mathbf{N}$, with the defining equations

$$B(0, a) = a + 1,$$
$$B(n + 1, a) = B(n, B(n, a)).$$

We formally define B as a term in $\mathrm{T}(;)$ by

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

From B we can define the exponential function $E := \lambda_n(Bn0)$ of type $E\colon \mathbf{N} \to \mathbf{N}$, and also iterated exponential functions like $\lambda_n(E(En))$.

Example: iteration

Now consider iteration $I(n, f) = f^n$, with f a variable of type $\mathbf{N} \multimap \mathbf{N}$.

$$\begin{aligned} I(0, f, a) &:= a, \\ I(n+1, f, a) &:= I(n, f, f(a)), \end{aligned} \quad \text{or} \quad \begin{aligned} I(0, f) &:= \mathrm{id}, \\ I(n+1, f) &:= I(n, f) \circ f. \end{aligned}$$

Formally, for every variable f of type $\mathbf{N} \multimap \mathbf{N}$ we have the term

$$I_f := \lambda_n \big(\mathcal{R}_{\mathbf{N} \multimap \mathbf{N}} n (\lambda_a a) \big(\lambda_{m,p,a} (p^{\mathbf{N} \multimap \mathbf{N}} (fa)) \big) \big).$$

For the general definition we need the pure safe types ρ_k, defined by $\rho_0 := \mathbf{N}$ and $\rho_{k+1} := \rho_k \multimap \rho_k$. Then within $\mathrm{T}(;)$ we can define

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

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

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

We must require that the value type is a safe type, for otherwise we could define

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

and $I_E(n,m) = E^n(m)$, a function of superelementary growth. We also need to require that the "previous"-variable is an output variable, because otherwise we could define

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

Then $S(n) = E^n(0)$.

Normalization

The size (or length) $|r|$ of a term r is the number of occurrences of constructors, variables and constants in r: $|x| = |C| = 1$, $|\lambda_n r| = |\lambda_z r| = |r| + 1$, and $|rs| = |r| + |s| + 1$.
In this section, the distinction between input and output variables and our two type formers $\longrightarrow$ and $\multimap$ plays no role.
We first deal with the (generalized) β-conversion rule above:

$$\big(\lambda_{\vec{x},x} r(\vec{x}, x)\big)\vec{s}s \mapsto \big(\lambda_{\vec{x}} r(\vec{x}, s)\big)\vec{s}.$$

A term is said to be in β-normal form if it does not contain a β-redex.
We want to show that every term reduces to a β-normal form.
This can be seen easily if we follow a certain order in our conversions. To define this order we have to make use of the fact that all our terms have types.

Normalization (continued)

A β-convertible term $\left(\lambda_{\vec{x},x} r(\vec{x}^{\vec{\rho}}, x^{\rho})\right)\vec{s}s$ is also called a cut with cut-type ρ. By the level of a cut we mean the level of its cut-type. The cut-rank of a term r is the least number bigger than the levels of all cuts in r. Now let t be a term of cut-rank $k+1$. Pick a cut of the maximal level k in t, such that s does not contain another cut of level k. (E.g., pick the rightmost cut of level k.) Then it is easy to see that replacing the picked occurrence of $(\lambda_{\vec{x},x} r(\vec{x}^{\vec{\rho}}, x^{\rho}))\vec{s}s$ in t by $(\lambda_{\vec{x}} r(\vec{x}, s))\vec{s}$ reduces the number of cuts of the maximal level k in t by 1. Hence

Theorem (β-Normalization)

We have an algorithm which reduces any given term into a β-normal form.

Normalization (continued)

Theorem (Upper bound for the complexity of β-normalization)

The β-normalization algorithm given in the proof above takes at most $E_k(l)$ steps to reduce a given term of cut-rank k and size l to normal form, where

$$E_0(l) := 0 \quad \textit{and} \quad E_{k+1}(l) := E_k(l) + l^{2^{E_k(l)}}.$$

We now show that we can also eliminate the recursion operator, and still have an elementary estimate on the time needed.

Lemma ($\mathcal{R}$ Elimination)

Let $t(\vec{x})$ be a β-normal term of safe type. There is an elementary function E_t such that: if $\vec{m}$ are safe type $\mathcal{R}$-free terms and the free variables of $t(\vec{m})$ are output variables of safe type, then in time $E_t(|\vec{m}|)$ (with $|\vec{m}| := \sum_i |m_i|$) one can compute an $\mathcal{R}$-free term $\mathrm{rf}(t; \vec{x}; \vec{m})$ such that $t(\vec{m}) \to^* \mathrm{rf}(t; \vec{x}; \vec{m})$.

Proof.

Induction on $|t|$

Normalization (continued)

Let the $\mathcal{R}$-rank of a term t be the least number bigger than the level of all value types τ of recursion operators $\mathcal{R}_\tau$ in t. By the rank of a term we mean the maximum of its cut-rank and its $\mathcal{R}$-rank.

Lemma

For every k there is an elementary function N_k such that every $\mathrm{T}(;)$-term t of rank $\leq k$ can be reduced in time $N_k(|t|)$ to $\beta\mathcal{R}$ normal form.

It remains to remove the occurrences of the cases operator $\mathcal{C}$. We may assume that only $\mathcal{C}_{\mathbf{N}}$ occurs.

Lemma ($\mathcal{C}$ Elimination)

Let t be an $\mathcal{R}$-free closed β-normal term of ground type $\mathbf{N}$. Then in time linear in $|t|$ one can reduce t to a numeral.

Theorem (Normalization)

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

Proof.

We produce an elementary function E_t such that for all numerals $\vec{n}$ with $t\vec{n}$ is of type $\mathbf{N}$ one can compute $\mathrm{nf}(t\vec{n})$ in time $E_t(|\vec{n}|)$. Let $\vec{x}$ be new variables such that $t\vec{x}$ is of type $\mathbf{N}$. The β normal form $\beta\text{-}\mathrm{nf}(t\vec{x})$ 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 $\vec{n}$.
By $\mathcal{R}$ Elimination one reduces to an $\mathcal{R}$-free term $\mathrm{rf}(\beta\text{-}\mathrm{nf}(t\vec{x}); \vec{x}; \vec{n})$ in time $F_t(|\vec{n}|)$ with F_t elementary. Since the running time bounds the size of the produced term, $|\mathrm{rf}(\beta\text{-}\mathrm{nf}(t\vec{x}); \vec{x}; \vec{n})| \leq F_t(|\vec{n}|)$. By a further β-normalization one can compute

$$\beta\mathcal{R}\text{-}\mathrm{nf}(t\vec{n}) = \beta\text{-}\mathrm{nf}(\mathrm{rf}(\beta\text{-}\mathrm{nf}(t\vec{x}); \vec{x}; \vec{n}))$$

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. □

Sufficiency

Conversely, we show that for every elementary function f there is a term t_f in $\mathbb{T}(;)$ of type $\mathbf{N}^{(k)} \to \mathbf{N}$ defining f.
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.

Recall that bounded minimization

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

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

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

Now the claim follows from the first examples above.

Future work

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

- $$\frac{\text{Arithmetic}}{\text{Gödel's T}} = \frac{\mathrm{A}(;)}{\mathrm{T}(;)} = \frac{\mathrm{LA}(;)}{\mathrm{LT}(;)}.$$
- Terms: Gödel's T over free algebras (possibly infinitary), 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.
- Further experiments with fine tuning of computational content: Uniform $\forall^{\mathsf{U}}$ and $\to^{\mathsf{U}}$.