Linear two-sorted arithmetic

Helmut Schwichtenberg

Mathematisches Institut, LMU, München

Kyoto University, Japan, 8. May 2009

Feasible computation with higher types

Gödel's T (1958) "Über eine bisher noch nicht benützte Erweiterung des finiten Standpunkts": finitely typed λ-terms with structural recursion.

$\mathrm{LT}(;)$ (Bellantoni, Niggl, S. 2000, 2002): restriction such that the definable functions are exactly the polynomial time computable ones.

Here:

$$\frac{\text{Heyting Arithmetic}}{\text{Gödel's } \mathrm{T}} = \frac{\mathrm{LA}(;)}{\mathrm{LT}(;)}$$

Related work

- Hofmann (1998): similar results with a very different proof technique. Ramification concepts have been considered earlier e.g. by Simmons (1988), Bellantoni & Cook (1992), Leivant & Marion (1998, 2001), and Pfenning (2001).
- The "tiered" typed λ-calculi of Leivant & Marion (1993) depend heavily on different representations of data (as words and as Church-like abstraction terms), which is not necessary in the $\mathrm{LT}(;)$-approach.
- Algebraic context semantics (Dal Lago 2006).
- Arai & Moser (2005), Beckmann & Weiermann (1996): Analysis (importance) of reduction strategies for $\mathcal{R}$.
- Baillot & Terui (2004): DLAL. Similar results; they notice that one can drop ! and have $\hookrightarrow$ and $\to$ instead.

Extending "Bellantoni/Cook" to higher types

- input $(\rho \hookrightarrow \sigma)$ $n, \bar{x}, \bar{y}, \dots$ (BC: normal)
  Recurse on.
  Use many times
- output $(\rho \to \sigma)$ $a, p, x, y, \dots$ (BC: safe)
  Cannot recurse on.
  Base type: use many times.
  Higher type: use at most once.

Typing of recursion

- Recursion with value type τ has type

$$\mathcal{R}_{\mathbf{N}}^{\tau} \colon \mathbf{N} \hookrightarrow \tau \to (\mathbf{N} \to \tau \to \tau) \hookrightarrow \tau \quad \text{with } \tau \text{ safe.}$$

- A type is safe if it does not contain the input arrow $\hookrightarrow$.

Terms

- built from (typed) input/output variables and constants by introduction and elimination for $\hookrightarrow$ and $\to$:

$$\bar{x}^\rho \mid x^\rho \mid C^\rho \quad \text{(constant)} \mid$$
$$(\lambda_{\bar{x}^\rho} r^\sigma)^{\rho\hookrightarrow\sigma} \mid (r^{\rho\hookrightarrow\sigma} s^\rho)^\sigma \quad (s \text{ input term: } \mathrm{FV}(s) \text{ input}) \mid$$
$$(\lambda_{x^\rho} r^\sigma)^{\rho\to\sigma} \mid (r^{\rho\to\sigma} s^\rho)^\sigma \quad (\text{higher type output vars in } r, s \text{ distinct}),$$

- The restriction on output variables in $r^{\rho\to\sigma} s$ ensures that every higher type output variable can occur at most once.
- A function f is definable in $\mathrm{LT}(;)$ if there is a closed term $t_f \colon \mathbf{N} \twoheadrightarrow \dots \mathbf{N} \twoheadrightarrow \mathbf{N}$ ($\twoheadrightarrow \in \{\hookrightarrow, \to\}$) denoting this function.

Numerals

Let $\mathbf{W} := \mathbf{L}(\mathbf{B})$, and

$$1 := \mathrm{nil}_{\mathbf{B}}, \qquad S_0 := \lambda_v(\mathtt{ff} :: v^{\mathbf{W}}), \qquad S_1 := \lambda_v(\mathtt{tt} :: v^{\mathbf{W}}).$$

Particular lists are $S_{i_1}(\dots(S_{i_n}1)\dots)$, called binary numerals (or words), denoted by $v, w \dots$.

Polynomials

- $\oplus \colon \mathbf{W} \hookrightarrow \mathbf{W} \to \mathbf{W}$ concatenates $\|v\|$ bits onto w:

  $$1 \oplus w = S_0 w, \qquad (S_i v) \oplus w = S_0(v \oplus w).$$

  The representing term is

  $$\bar{v} \oplus w := \mathcal{R}_{\mathbf{W}\to\mathbf{W}} \bar{v} S_0 \lambda_{\_,\_,p,w}(S_0(p^{\mathbf{W}\to\mathbf{W}} w))w.$$

- $\odot \colon \mathbf{W} \hookrightarrow \mathbf{W} \hookrightarrow \mathbf{W}$ has output length $\|v\| \cdot \|w\|$:

  $$v \odot 1 = v, \qquad v \odot (S_i w) = v \oplus (v \odot w).$$

  The representing term is

  $$\bar{v} \odot \bar{w} := \mathcal{R}_{\mathbf{W}} \bar{w} \bar{v} \lambda_{\_,\_,p}(\bar{v} \oplus p).$$

A non-example: exponentiation

- Notice that $\oplus\colon \mathbf{W} \hookrightarrow \mathbf{W} \to \mathbf{W}$, and the value type for the recursion was $\mathbf{W} \to \mathbf{W}$, which is safe.
- If we try to go on and define exponentiation from multiplication just as $\odot$ was defined from $\oplus$, we find that we cannot go ahead, because of the different typing $\odot\colon \mathbf{W} \hookrightarrow \mathbf{W} \hookrightarrow \mathbf{W}$.

Two recursions

Consider

$$D(1) := S_0(1), \qquad E(1) := 1,$$
$$D(S_i(w)) := S_0(S_0(D(w))), \qquad E(S_i(w)) := D(E(w)).$$

The corresponding terms are

$$D := \lambda_{\bar{w}}(\mathcal{R}_{\mathbf{W}}\bar{w}(S_0 1)\lambda_{\_,\_,p}(S_0(S_0 p))), \qquad E := \lambda_{\bar{w}}(\mathcal{R}_{\mathbf{W}}\bar{w}1\lambda_{\_,\_,p}(Dp)).$$

Here D is legal, but E is not: the application Dp is not allowed.

Recursion with parameter substitution

Consider

$$\begin{aligned} E(1,v) &:= S_0(v), \\ E(S_i(w),v) &:= E(w, E(w,v)), \end{aligned} \quad \text{or} \quad \begin{aligned} E(1) &:= S_0, \\ E(S_i(w)) &:= E(w) \circ E(w). \end{aligned}$$

The corresponding term

$$\lambda_{\bar{w}}(\mathcal{R}_{\mathbf{W}\to\mathbf{W}}\bar{w}S_0\lambda_{\_,\_,p,v}(p^{\mathbf{W}\to\mathbf{W}}(pv)))$$

does not satisfy the linearity condition: the higher type variable p occurs twice, and the typing of $\mathcal{R}$ requires p to be an output variable.

Higher argument types

- Consider iteration $I(n, f) = f^n$:

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

  It is definable by a term with f a parameter of type $\mathbf{W} \to \mathbf{W}$:

$$I_f := \lambda_n(\mathcal{R}_{\mathbf{W}\to\mathbf{W}} n(\lambda_w w)\lambda_{\_,p,w}(p^{\mathbf{W}\to\mathbf{W}}(fw))).$$

- f must be an input variable, because the step argument of a recursion is by definition an input argument. Thus $\lambda_f I_f$ may only be applied to input terms of type $\mathbf{W} \to \mathbf{W}$.
- We cannot define the exponential function by

$$\lambda_n(\mathcal{R}_{\mathbf{W}\to\mathbf{W}} nS\lambda_{\_,p}(I_p 2)).$$

  The step type requires p to be an output variable, but I_p is only correctly formed if p is an input variable.

Normalization

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

- Let $\vec{z}$ be new variables. Compute the normal form of $t\vec{z}$ (needs constant time w.r.t. $\vec{n}$).
- $\mathrm{nf}(t\vec{z})$ is "simple" (i.e., no free or bound higher type input variables).
- Reduce to an $\mathcal{R}$-free simple term $\mathrm{rf}(\mathrm{nf}(t\vec{z}); \vec{z}; \vec{n})$ 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, $\|\mathrm{rf}(\mathrm{nf}(t\vec{z}); \vec{z}; \vec{n})\| \leq P_t(\|\vec{n}\|)$.
- By Sharing Normalization one computes $\mathrm{nf}(t\vec{n}) = \mathrm{nf}(\mathrm{rf}(\mathrm{nf}(t\vec{z}); \vec{z}; \vec{n}))$ in time $O(P_t(\|\vec{n}\|)^2)$.

Linear two-sorted arithmetic $\mathrm{LA}(;)$

Using the Curry-Howard correspondence, we transfer the term system $\mathrm{LT}(;)$ to a logical system $\mathrm{LA}(;)$ of arithmetic, with

- two arrow types, $\rho \hookrightarrow \sigma$ (input) and $\rho \to \sigma$ (output),
- two sorts of variables, input ones $\bar{x}$ and output ones x, and
- two implications, $A \hookrightarrow B$ (input) and $A \to B$ (output).

Restrictions:

- Proofs of the premise of $A \hookrightarrow B$ are only allowed to use input assumptions or input variables.
- Proofs of the premise of $A \to B$ can only have at most one use of the hypothesis, in case its type is not a base type.

Double use of assumptions

- Consider

$$E(1,y) := S_0(y), \quad E(S_i(x),y) := E(x,E(x,y)), \qquad \text{or} \qquad E(1) := S_0, \quad E(S_i(x)) := E(x) \circ E(x).$$

  Then $E(x) = S_0^{(2^{\|x\|-1})}$, i.e., E grows exponentially.
- Corresponding existence proof. Show by induction on x

$$\forall_{x,y} \exists_v (\|v\| = 2^{\|x\|-1} + \|y\|).$$

- Double use of the ("functional") induction hypothesis is responsible for exponential growth. The linearity restriction on output implications will exclude such proofs.

Substitution in function parameters

- Consider the iteration functional $I(x, f) = f^{(\|x\|-1)}$; it is considered feasible. However, substituting doubling D with $\|D(x)\| = 2\|x\|$ yields $I(x, D) = D^{(\|x\|-1)}$.
- The corresponding proofs of

$$\forall_x(\forall_y \exists_z(\|z\| = 2\|y\|) \to \forall_y \exists_v(\|v\| = 2^{\|x\|-1} + \|y\|)), \qquad (1)$$

$$\forall_y \exists_z(\|z\| = 2\|y\|) \qquad (2)$$

  are unproblematic, but we need to forbid a cut here.
- Solution: ramification concept. (2) is proved by induction on y, hence needs: $\forall_{\bar{y}} \exists_z(\|z\| = 2\|\bar{y}\|)$. Cut excluded by ramification condition: the "kernel" of (1) – to be proved by induction on x – is safe and hence does not contain such universal subformulas proved by induction.

Iterated induction

- It might seem that our restrictions are so tight that they rule out any form of nested induction.
- However, this is not true. One can define, e.g., (a form of) multiplication on top of addition: First one proves

$$\forall_{\bar{x}} \forall_{y} \exists_{z} (\|z\| = \|\bar{x}\| + \|y\|)$$

by induction on $\bar{x}$, and then

$$\forall_{\bar{y}} \exists_{z} (\|z\| = \|\bar{x}\| \cdot \|\bar{y}\|)$$

by induction on $\bar{y}$ with a parameter $\bar{x}$.

Linear two-sorted arithmetic $\mathrm{LA}(;)$

- $\mathrm{LA}(;)$-formulas are

$$I(\vec{r}\,) \mid A \hookrightarrow B \mid A \to B \mid \forall_{\bar{x}^\rho} A \mid \forall_{x^\rho} A \qquad (\vec{r} \text{ terms from } \mathrm{T}).$$

- Define falsity $\mathbf{F}$ by $\mathrm{Eq}(\mathrm{ff}, \mathrm{tt})$ and $\neg A$ by $A \to \mathbf{F}$.
- Define $\tau(A)$ by

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

- A is safe if $\tau(A)$ is safe, i.e., $\hookrightarrow$-free.

Linear two-sorted arithmetic $\mathrm{LA}(;)$ (ctd.)

- The induction axiom for **N** is

$$\mathrm{Ind}_{n,A} \colon \forall_n (A(0) \to \forall_a (A(a) \to A(\mathrm{S}a)) \hookrightarrow A(n^{\mathbf{N}}))$$

  with n an input and a an output variable, and A safe.
- It has the type of the recursion operator which will realize it:

$$\mathbf{N} \hookrightarrow \tau \to (\mathbf{N} \to \tau \to \tau) \hookrightarrow \tau \quad \text{where } \tau = \tau(A) \text{ is safe.}$$

Ordinary proof terms

are built from axioms, assumption and object terms by the usual rules for both implications ($\hookrightarrow$ and $\to$) and both universal quantifiers (over input and output variables):

$$
\begin{aligned}
& c^A \quad \text{(axiom)} \mid \\
& \bar{u}^A, u^A \quad \text{(input and output assumption variables)} \mid \\
& (\lambda_{\bar{u}^A} M^B)^{A \hookrightarrow B} \mid (M^{A \hookrightarrow B} N^A)^B \mid \\
& (\lambda_{u^A} M^B)^{A \to B} \mid (M^{A \to B} N^A)^B \mid \\
& (\lambda_{\bar{x}^\rho} M^A)^{\forall_{\bar{x}} A} \mid (M^{\forall_{\bar{x}^\rho} A(\bar{x})} r^\rho)^{A(r)} \mid \\
& (\lambda_{x^\rho} M^A)^{\forall_x A} \mid (M^{\forall_{x^\rho} A(x)} r^\rho)^{A(r)}
\end{aligned}
$$

with r a term in T, not necessarily in $\mathrm{LT}(;)$.

$\mathrm{LA}(;)$-proof terms M and $\mathrm{CV}(M)$

are defined simultaneously:

- If $\tau(A) = \varepsilon$, then every ordinary proof term M^A is an $\mathrm{LA}(;)$-proof term; $\mathrm{CV}(M) := \emptyset$.
- $(M^{A \hookrightarrow B} N^A)^B$, if all variables in $\mathrm{CV}(N)$ are input.
- $(M^{A \to B} N^A)^B$, if the higher type output variables in $\mathrm{CV}(M)$ and $\mathrm{CV}(N)$ are disjoint.
- $(M^{\forall_{\bar{x}} A(\bar{x})} r)^{A(r)}$ if r is an input $\mathrm{LT}(;)$-term.
- If $(M^{\forall_x A(x)} r)$ if r is an $\mathrm{LT}(;)$-term, and the higher type output variables in $\mathrm{CV}(M)$ are not free in r.

$\mathrm{LA}(;)$ and its provably recursive functions

- A k-ary numerical function f is provably recursive in $\mathrm{LA}(;)$ if there is a Σ_1-formula $G_f(n_1, \dots, n_k, a)$ denoting the graph of f, and a derivation M_f in $\mathrm{LA}(;)$ of

  $$\forall_{n_1,\dots,n_k} \exists_a G_f(n_1, \dots, n_k, a).$$

  (n_i input and a output variables of type $\mathbf{W}$).
- The functions provably recursive in $\mathrm{LA}(;)$ are exactly the definable functions of $\mathrm{LT}(;)$ of type $\mathbf{W}^k \hookrightarrow \mathbf{W}$ (i.e., the ones computable in polynomial time).

Example: Insertion sort in $\mathrm{LA}(;)$

- Goal: the insertion sort algorithm is the computational content of an appropriate proof.
- Let I insert a into a list l, in the first place where it finds an element bigger:

$$\mathrm{I}(a, \mathrm{nil}) := a :: \mathrm{nil}, \qquad \mathrm{I}(a, b :: l) := \begin{cases} a :: b :: l & \text{if } a \leq b, \\ b :: \mathrm{I}(a, l) & \text{otherwise} \end{cases}$$

- Using I, define a function S sorting a list l:

$$S(\mathrm{nil}) := \mathrm{nil}, \qquad S(a :: l) := \mathrm{I}(a, S(l)).$$

- Represent I, S by inductive definitions of their graphs.

Example: Insertion sort in $\mathrm{LA}(;)$ (ctd.)

Want to derive $\exists_{l'} S(l, l')$ in $\mathrm{LA}(;)$. However, we cannot do this. All we can achieve is

$$\mathrm{lh}(l) \leq n \to \exists_{l'} S(l, l') \quad \text{for any input parameter } n.$$

In more detail, prove

- $\forall_{a,l,n} \forall_{i \leq n} \exists_{l'} \mathrm{I}(a, \mathrm{tl}_{\min(i,\mathrm{lh}(l))}(l), l')$, by induction on n.
- $\forall_{l,n,m}(m \leq n \to \exists_{l'} S(\mathrm{tl}_{\min(m,\mathrm{lh}(l))}(l), l'))$, by induction on m.
- Specializing this to l, n, n we obtain $\mathrm{lh}(l) \leq n \to \exists_{l'} S(l, l')$.

References

- S. Bellantoni, K.-H. Niggl and H.S., Higher type recursion, ramification and polynomial time. APAL 104 (2000) 17–30.
- H.S. and S. Bellantoni, Feasible computation with higher types. In: Proc. MOD 2002 (Kluwer) 399–415.
- H.S., An arithmetic for polynomial-time computation. TCS 357 (2006) 202–214.