Linear two-sorted constructive arithmetic

Helmut Schwichtenberg

Mathematisches Institut, LMU, München

Computability Theory and Foundations of Mathematics,
Waseda University, Tokyo, 20. & 21. September 2016

Feasible computation with higher types

Gödel's T (1958): finitely typed λ-terms with structural recursion.

$\mathrm{LT}(;)$ (linear two-sorted λ-terms) restricts T s.t. that the definable functions are the polynomial time (ptime) computable ones. $\mathrm{LT}(;)$ generalizes Bellantoni & Cook (1992) to finite types.

$\mathrm{LA}(;)$ solves

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

Its provably recursive functions are the ptime computable ones.

Problem: how to cover ptime algorithms (not only functions), e.g. divide-and-conquer ones (quicksort, treesort): they are not linear.

Sources of exponential complexity. (i) Two recursions

We define a function D doubling a natural number and – using D – a function $E(n)$ representing 2^n:

$$\begin{aligned} D(0) &:= 0, & E(0) &:= 1, \\ D(S(n)) &:= S(S(D(n))), & E(S(n)) &:= D(E(n)). \end{aligned}$$

Problem: previous value $E(n)$ taken as recursion argument for D.
Cure: mark argument positions in arrow types as input or output.
Recursion arguments are always input positions.

(ii) Double use of higher type values

Define F as the 2^n-th iterate of D:

$$\begin{aligned} F(0, m) &:= D(m), \\ F(S(n), m) &:= F(n, F(n, m)) \end{aligned} \qquad \text{or} \qquad \begin{aligned} F(0) &:= D, \\ F(S(n)) &:= F(n) \circ F(n). \end{aligned}$$

Problem: in the recursion equation previous value is used twice.
Cure: linearity restriction. No double use of higher type output.

(iii) Marked value types

Define $I(n, f)$ as the n-th iterate f^n of f. Thus $I(n, D)(m) = 2^n m$.

$$\begin{aligned} I(0, f, m) &:= m, \\ I(S(n), f, m) &:= f(I(n, f, m)) \end{aligned} \qquad \text{or} \qquad \begin{aligned} I(0, f) &:= \mathrm{id}, \\ I(S(n), f) &:= f \circ I(n, f). \end{aligned}$$

Problem: since $D \colon \mathbf{N} \hookrightarrow \mathbf{N}$, I's value type is $(\mathbf{N} \hookrightarrow \mathbf{N}) \to \mathbf{N} \hookrightarrow \mathbf{N}$.
Cure: only allow "safe" types as value types of a recursion (no marked argument positions).

(I will be admitted is our setting. This is not the case in Cook and Kapron's PV^ω, since PV^ω is closed under substitution.)

Linear two-sorted terms

Types with input arrow $\hookrightarrow$ and output arrow $\to$:

$$\rho, \sigma ::= \iota \mid \rho \hookrightarrow \sigma \mid \rho \to \sigma \quad \text{with } \iota \text{ base type } (\mathbf{B}, \mathbf{N}, \rho \times \sigma, \mathbf{L}(\rho)).$$

ρ is safe if it does not involve the input arrow $\hookrightarrow$.
Input variables $\bar{x}^\rho$ and output variables x^ρ (typed).
Constants are (i) constructors, (ii) recursion operators

$$\left.\begin{aligned} \mathcal{R}^\tau_{\mathbf{N}} &: \mathbf{N} \hookrightarrow \tau \to (\mathbf{N} \hookrightarrow \tau \to \tau) \hookrightarrow \tau \\ \mathcal{R}^\tau_{\mathbf{L}(\rho)} &: \mathbf{L}(\rho) \hookrightarrow \tau \to (\rho \hookrightarrow \mathbf{L}(\rho) \hookrightarrow \tau \to \tau) \hookrightarrow \tau \end{aligned}\right. \quad (\tau \text{ safe}),$$

and (iii) cases operators (τ safe)

$$\begin{aligned} \mathcal{C}^\tau_{\mathbf{N}} &: \mathbf{N} \to \tau \to (\mathbf{N} \hookrightarrow \tau) \to \tau, \\ \mathcal{C}^\tau_{\mathbf{L}(\rho)} &: \mathbf{L}(\rho) \to \tau \to (\rho \hookrightarrow \mathbf{L}(\rho) \hookrightarrow \tau) \to \tau, \\ \mathcal{C}^\tau_{\rho \times \sigma} &: \rho \times \sigma \to (\rho \hookrightarrow \sigma \hookrightarrow \tau) \to \tau. \end{aligned}$$

$\mathrm{LT}(;)$-terms built from variables and constants by introduction and elimination rules for the two type forms $\rho \hookrightarrow \sigma$ and $\rho \to \sigma$:

$$
\begin{array}{ll}
\bar{x}^\rho \mid x^\rho \mid C^\rho \text{ (constant)} \mid & \\
(\lambda_{\bar{x}^\rho} r^\sigma)^{\rho \hookrightarrow \sigma} \mid (r^{\rho \hookrightarrow \sigma} s^\rho)^\sigma & (s \text{ an input term}) \mid \\
(\lambda_{x^\rho} r^\sigma)^{\rho \to \sigma} \mid (r^{\rho \to \sigma} s^\rho)^\sigma & (\text{higher type output vars in } r, s \text{ distinct}, \\
& r \text{ does not start with } \mathcal{C}_\iota^\tau) \mid \\
\mathcal{C}_\iota^\tau t \vec{r} & (\text{h.t. output vars in } \mathrm{FV}(t) \text{ not in } \vec{r})
\end{array}
$$

with as many r_i as there are constructors of ι. s is an input term if

- all its free variables are input variables, or else
- s is of higher type and all its higher type free variables are input variables.

The parse dag computation model

Represent terms as directed acyclic graphs (dag), where only nodes for terms of base type can have in-degree > 1. Nodes can be

- terminal nodes labelled by a variable or constant,
- abstraction nodes with 1 successor, labelled with an (input or output) variable and a pointer to the successor node, or
- application nodes with 2 successors, labelled with 2 pointers.

A parse dag is a parse tree for a term.

The treesort algorithm

$$
\begin{aligned}
\mathrm{TreeSort}(l) &= \mathrm{Flatten}(\mathrm{MakeTree}(l)),\\[1ex]
\mathrm{MakeTree}([]) &= \diamond,\\
\mathrm{MakeTree}(a :: l) &= \mathrm{Insert}(a, \mathrm{MakeTree}(l)),\\[1ex]
\mathrm{Insert}(a, \diamond) &= C_a(\diamond, \diamond),\\
\mathrm{Insert}(a, C_b(u, v)) &= \begin{cases} C_b(\mathrm{Insert}(a, u), v) & \text{if } a \leq b \\ C_b(u, \mathrm{Insert}(a, v)) & \text{if } b < a, \end{cases}\\[1ex]
\mathrm{Flatten}(\diamond) &= [],\\
\mathrm{Flatten}(C_b(u, v)) &= \mathrm{Flatten}(u) * (b :: \mathrm{Flatten}(v)).
\end{aligned}
$$

Problem: two recursive calls in $\mathrm{Flatten}$, not allowed in $\mathrm{LT}(;)$.
Cure: analysis of $\mathrm{Flatten}$ in the parse dag computation model.

We estimate the number $\#t$ of steps it takes to reduce a term t to its normal form $\mathrm{nf}(t)$.

Lemma. Let l be a numeral of type $\mathbf{L}(\mathbf{N})$. Then $\#(l * l') = O(|l|)$.

For $\#\mathrm{Flatten}(u)$ use this size function for numerals u of type $\mathbf{T}$:

$$\begin{aligned} \|\diamond\| &:= 0, \\ \|C_a(u, v)\| &:= 2\|u\| + \|v\| + 3. \end{aligned}$$

Lemma. Let u be a numeral of type $\mathbf{T}$. Then

$$\#\mathrm{Flatten}(u) = O(\|u\|).$$

Goal: all functions definable in $\mathrm{LT}(;) + \mathrm{Flatten}$ are polytime computable. Call a term

- $\mathcal{RD}$-free: no recursion constant $\mathcal{R}$, no $\mathrm{Flatten}$.
- simple: no higher type input variables.

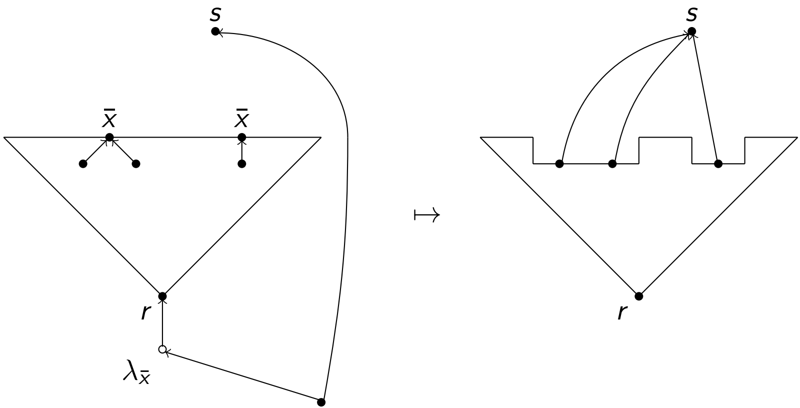
Lemma (Sharing normalization)

Let t be an $\mathcal{RD}$-free simple term. Then a parse dag for $\mathrm{nf}(t)$, of size at most $\|t\|$, can be computed from t in time $O(\|t\|^2)$.

Corollary (Base normalization)

Let t be a closed $\mathcal{RD}$-free simple term of type $\mathbf{N}$ or $\mathbf{L}(\mathbf{N})$. Then $\mathrm{nf}(t)$ can be computed from t in time $O(\|t\|^2)$, and $\|\mathrm{nf}(t)\| \leq \|t\|$.

$(\lambda_{\bar{x}} r(\bar{x}))s$ with $\bar{x}$ of base type



Lemma ($\mathcal{RD}$-elimination)

Let $t(\vec{x})$ be a simple term of safe type. There is a polynomial P_t such that: if $\vec{r}$ are safe type $\mathcal{RD}$-free closed simple terms and the free variables of $t(\vec{r})$ are output variables, then in time $P_t(\|\vec{r}\|)$ one can compute an $\mathcal{RD}$-free simple term $\mathrm{rdf}(t;\vec{x};\vec{r})$ such that $t(\vec{r}) \to^* \mathrm{rdf}(t;\vec{x};\vec{r})$.

Proof.

By induction on $\|t\|$ (cf. Chapter 8 of H.S. & S.Wainer, Proofs and Computations, 2012). Need an additional case for $\mathrm{Flatten}$, and $\#\mathrm{Flatten}(u) = O(\|u\|)$. $\square$

Theorem (Normalization)

Let $t\colon \mathbf{N} \twoheadrightarrow \ldots \mathbf{N} \twoheadrightarrow \mathbf{N}$ (with $\twoheadrightarrow \in \{\hookrightarrow, \to\}$) be a closed term in $\mathrm{LT}(;) + \mathrm{Flatten}$. Then t denotes a polytime function.

Conclusion

- $\mathrm{LA}(;) \sim \mathrm{LT}(;)$ via Curry-Howard correspondence.
- $$\frac{\text{Heyting Arithmetic}}{\text{Gödel's T}} = \frac{\mathrm{LA}(;)}{\mathrm{LT}(;)} = \frac{\mathrm{LA}(;) + \mathrm{Flatten}}{\mathrm{LT}(;) + \mathrm{Flatten}}$$
- $\mathrm{LA}(;) + \mathrm{Flatten} \vdash \forall_{l,\bar{n}}(|l| \leq \bar{n} \to \exists_u S(l, u))$
- Computational content of this proof: $(\mathrm{LT}(;) + \mathrm{Flatten})$-term. Can be extracted by realizability. $\sim$ treesort algorithm.