

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.

LT(;) (**linear two-sorted** λ -terms) restricts T s.t. that the definable functions are the polynomial time (ptime) computable ones.

LA(;) solves

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

Its provably recursive functions are the ptime computable ones.

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

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 :

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

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 $I(n, D)(m) = 2^n m$.

$$\begin{array}{ll} I(0, f, m) := m, & \\ I(S(n), f, m) := f(I(n, f, m)) & \text{or} \quad I(0, f) := \text{id}, \\ & I(S(n), f) := f \circ I(n, f). \end{array}$$

Problem: since $D: \mathbf{N} \hookrightarrow \mathbf{N}$, I needs type $(\mathbf{N} \hookrightarrow \mathbf{N}) \rightarrow \mathbf{N} \hookrightarrow \mathbf{N}$.

Cure: only allow “safe” types as value types of a recursion (no marked argument positions).

(I will be admitted in 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** $\rightarrow$:

$\rho, \sigma ::= \iota \mid \rho \hookrightarrow \sigma \mid \rho \rightarrow \sigma$ with ι 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

$$\begin{aligned} \mathcal{R}_{\mathbf{N}}^\tau &: \mathbf{N} \hookrightarrow \tau \rightarrow (\mathbf{N} \hookrightarrow \tau \rightarrow \tau) \hookrightarrow \tau \\ \mathcal{R}_{\mathbf{L}(\rho)}^\tau &: \mathbf{L}(\rho) \hookrightarrow \tau \rightarrow (\rho \hookrightarrow \mathbf{L}(\rho) \hookrightarrow \tau \rightarrow \tau) \hookrightarrow \tau \quad (\tau \text{ safe}), \end{aligned}$$

and (iii) cases operators (τ safe)

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

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

$$\begin{aligned}
 & \bar{x}^\rho \mid x^\rho \mid C^\rho \text{ (constant)} \mid \\
 & (\lambda_{\bar{x}^\rho} r^\sigma)^{\rho \hookrightarrow \sigma} \mid (r^{\rho \hookrightarrow \sigma} s^\rho)^\sigma \text{ (} s \text{ an input term)} \mid \\
 & (\lambda_{x^\rho} r^\sigma)^{\rho \rightarrow \sigma} \mid (r^{\rho \rightarrow \sigma} s^\rho)^\sigma \text{ (higher type output vars in } r, s \text{ distinct,} \\
 & \hspace{15em} r \text{ does not start with } C_\iota^\tau) \mid \\
 & C_\iota^\tau t \vec{r} \hspace{15em} \text{(h.t. output vars in } FV(t) \text{ not in } \vec{r})
 \end{aligned}$$

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

$$\text{TreeSort}(l) = \text{Flatten}(\text{MakeTree}(l)),$$

$$\text{MakeTree}([]) = \diamond,$$

$$\text{MakeTree}(a :: l) = \text{Insert}(a, \text{MakeTree}(l)),$$

$$\text{Insert}(a, \diamond) = C_a(\diamond, \diamond),$$

$$\text{Insert}(a, C_b(u, v)) = \begin{cases} C_b(\text{Insert}(a, u), v) & \text{if } a \leq b \\ C_b(u, \text{Insert}(a, v)) & \text{if } b < a, \end{cases}$$

$$\text{Flatten}(\diamond) = [],$$

$$\text{Flatten}(C_b(u, v)) = \text{Flatten}(u) * (b :: \text{Flatten}(v)).$$

Problem: two recursive calls in Flatten, not allowed in LT(;).

Cure: analysis of 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 $\text{nf}(t)$.

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

For $\#\text{Flatten}(u)$ we use a 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

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

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

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

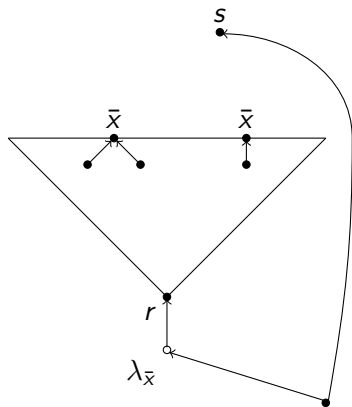
Lemma (Sharing normalization)

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

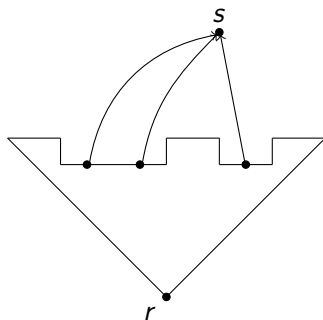
Corollary (Base normalization)

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

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



$\mapsto$



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 $\text{rdf}(t; \vec{x}; \vec{r})$ such that $t(\vec{r}) \rightarrow^ \text{rdf}(t; \vec{x}; \vec{r})$.*

Proof.

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

Theorem (Normalization)

Let $t: \mathbf{N} \twoheadrightarrow \dots \mathbf{N} \twoheadrightarrow \mathbf{N}$ (with $\twoheadrightarrow \in \{\hookrightarrow, \rightarrow\}$) be a closed term in $\text{LT}(;) + \text{Flatten}$. Then t denotes a polytime function.

Conclusion

- ▶ $\text{LA}(\cdot) \sim \text{LT}(\cdot)$ via Curry-Howard correspondence.



$$\frac{\text{Heyting Arithmetic}}{\text{Gödel's T}} = \frac{\text{LA}(\cdot)}{\text{LT}(\cdot)} = \frac{\text{LA}(\cdot) + \text{Flatten}}{\text{LT}(\cdot) + \text{Flatten}}$$

- ▶ $\text{LA}(\cdot) + \text{Flatten} \vdash \forall_{l, \bar{n}} (|l| \leq \bar{n} \rightarrow \exists_u S(l, u))$
- ▶ Computational content of this proof ($(\text{LA}(\cdot) + \text{Flatten})$ -term extracted from it by realizability) $\sim$ treesort algorithm.