Linear two-sorted constructive arithmetic

Helmut Schwichtenberg

Mathematisches Institut, LMU, München

Dipartimento di Informatica, Università degli Studi di Verona,
March 15, 2016

- Proofs may have computational content, which can be extracted (via realizability).
- Proofs (but not programs) can be checked for correctness.

Issues:

- Need to extend classical to constructive logic.
- Complexity.

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.

$\mathrm{LA}(;)$ 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 (like quicksort, treesort).

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

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

Constructive logic

- Use $\to$, $\forall$ only, defined by introduction and elimination rules.
- View $\exists_x A$, $A \vee B$, $A \wedge B$ as inductively defined predicates (with parameters A, B).
- In addition, define classical existence and disjunction by

$$\tilde{\exists}_x A := \neg \forall_x \neg A,$$
$$A \mathbin{\tilde{\vee}} B := \neg(\neg A \wedge \neg B)$$

where $\neg A := (A \to \mathbf{F})$ and $\mathbf{F} := (0 = 1)$.

Proof terms: assumptions variables, $\to$-rules

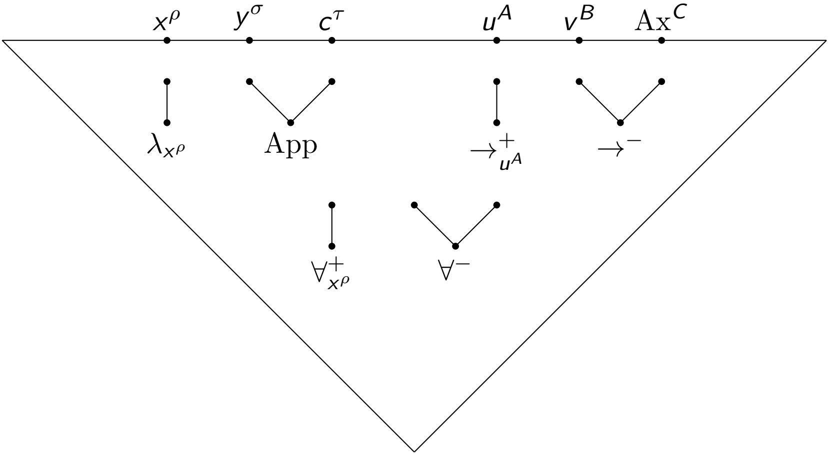
Assumption variables: $u\colon A$ (or u^A)

Derivation	Term
$$\dfrac{\begin{matrix}[u\colon A] \\ \mid M \\ B\end{matrix}}{A \to B}\ \to^+ u$$	$(\lambda_{u^A} M^B)^{A \to B}$
$$\dfrac{\begin{matrix}\mid M \\ A \to B\end{matrix} \qquad \begin{matrix}\mid N \\ A\end{matrix}}{B}\ \to^-$$	$(M^{A \to B} N^A)^B$

Proof terms: ∀-rules

Derivation	Term
$\dfrac{\begin{array}{c} \mid M \\ A \end{array}}{\forall_x A}\,\forall^+ x \quad \text{(var. cond.)}$	$(\lambda_x M^A)^{\forall_x A}$ (var. cond.)
$\dfrac{\begin{array}{c} \mid M \\ \forall_x A(x) \end{array} \qquad r}{A(r)}\,\forall^-$	$(M^{\forall_x A(x)} r)^{A(r)}$

Proof terms in natural deduction



The realizability interpretation transforms such a proof term directly into an object term.

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:

$$D(0) := 0, \qquad E(0) := 1,$$
$$D(S(n)) := S(S(D(n))), \qquad E(S(n)) := D(E(n)).$$

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:

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

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

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

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

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

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

$$\begin{aligned} \mathcal{R}^\tau_{\mathbf{N}} &\colon \mathbf{N} \hookrightarrow \tau \to (\mathbf{N} \hookrightarrow \tau \to \tau) \hookrightarrow \tau \\ \mathcal{R}^\tau_{\mathbf{L}(\rho)} &\colon \mathbf{L}(\rho) \hookrightarrow \tau \to (\rho \hookrightarrow \mathbf{L}(\rho) \hookrightarrow \tau \to \tau) \hookrightarrow \tau \end{aligned} \qquad (\tau \text{ safe}),$$

and (iii) cases operators (τ safe)

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

$\mathrm{LT}(;)$-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 size $\|d\|$ of a parse dag d is the number of nodes in it.
- A parse dag is conformal if (i) every node with in-degree greater than 1 is of base type, and (ii) every maximal path to a bound variable x passes through the same binding λ_x-node.
- A parse dag is h-affine if every higher type variable occurs at most once in the dag, except in the alternatives of a cases operator.

We identify a parse dag with the term it represents.

Steps requiring 1 time unit:

- Creation of a node given its label and pointers to successors.
- Deletion of a node.
- Given a pointer to an interior node, to obtain a pointer to one of its successors.
- Test on the type and the label of a node, and on the variable or constant in case the node is terminal.

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 $\#\mathrm{Flatten}(u)$ 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

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

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

- $\mathcal{RD}$-free if it contains neither recursion constants $\mathcal{R}$ nor $\mathrm{Flatten}$, and
- simple if it contains no higher type input variables.

Simple terms closed under reduction, subterms, application.

Lemma (Simplicity)

Let t be a base type term whose free variables are of base type. Then $\mathrm{nf}(t)$ is simple.

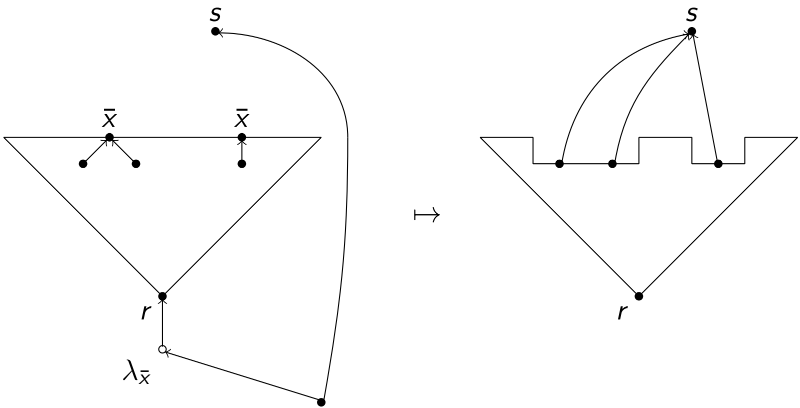
Lemma (Sharing normalization)

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

Theorem (Normalization)

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

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 $\tau(A)$ 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

$$\mathrm{Ind}_{\bar{n},A} \colon \forall_{\bar{n}}(A(0) \to \forall_{\bar{m}}(A(\bar{m}) \to A(S\bar{m})) \hookrightarrow A(\bar{n}^{\mathbf{N}}))$$

  with A safe.

- It has the type of the recursion operator which will realize it:

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

Treesort in $\mathrm{LA}(;) + \mathrm{Flatten}$

A tree u is sorted if the list $\mathrm{Flatten}(u)$ is sorted. We recursively define a function I inserting an element a into a tree u such that, if u is sorted, then so is $\mathrm{I}(a, u)$:

$$\mathrm{I}(a, \diamond) := C_a(\diamond, \diamond),$$

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

and, using I, a function S sorting a list l into a tree:

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

We represent I, S by (n.c.) inductive definitions of their graphs. Write $\mathrm{I}(a, u, u')$ for $\mathrm{I}(a, u) = u'$ and $S(l, u)$ for $S(l) = u$. Clauses:

$$\begin{aligned}
&\mathrm{I}(a, \diamond, C_a(\diamond, \diamond)),\\
&a \leq b \to \mathrm{I}(a, u, u') \to \mathrm{I}(a, C_b(u, v), C_b(u', v)),\\
&b < a \to \mathrm{I}(a, v, v') \to \mathrm{I}(a, C_b(u, v), C_b(u, v')),
\end{aligned}$$

$$\begin{aligned}
&S([], \diamond),\\
&S(l, u) \to \mathrm{I}(a, u, u') \to S(a :: l, u').
\end{aligned}$$

- We would like to derive $\exists_u S(l, u)$ in $\mathrm{LA}(;) + \mathrm{Flatten}$.
- However, this is not possible.
- All we can get is $|l| \leq n \to \exists_u S(l, u)$ (n an input parameter).

Lemma (Tree insertion)
$\forall_{a,n,u}(|u| \leq n \to \exists_{u'} \mathrm{I}(a, u, u'))$.

Proof. Fix a. Do induction on n.

Let $\mathrm{tl}_i(l)$ be the tail of the list l of length i, if $i < |l|$, and l else.

Lemma (Treesort)
$\forall_{l,n,m}(m \leq n \to \exists_u S(\mathrm{tl}_{\min(m,|l|)}(l), u))$.

Proof. Fix l, n. Do induction on m.

Extraction from tree insertion lemma

```
[a,n](Rec nat=>bbin=>bbin)n([u]C a Emp Emp)
 ([n1,h,u][if (Ht u<=n1)
              (h u)
              [if (a<=Lb u)
                  (C Lb u(h L u)R u)
                  (C Lb u L u(h R u))]])
```

Represents the function f of type $\mathbf{N} \to \mathbf{N} \hookrightarrow \mathbf{T} \to \mathbf{T}$ defined by

$$f(a, 0, u) := C_a(\diamond, \diamond),$$

$$f(a, n+1, u) := \begin{cases} f(a, n, u) & \text{if } |u| \leq n, \\ C_{\mathrm{Lb}(u)}(f(a, n, L(u)), R(u)) & \text{if } n < |u|,\ a \leq \mathrm{Lb}(u), \\ C_{\mathrm{Lb}(u)}(L(u), f(a, n, R(u))) & \text{if } n < |u|,\ \mathrm{Lb}(u) < a \end{cases}$$

with $\mathrm{Lb}(u), L(u), R(u)$ label and left and right subtree of $u \neq \diamond$.

Extraction from treesort lemma

```
[l,n,m](Rec nat=>bbin)m Emp
 ([m1,u][if (Lh l<=m1)
   u
   [if m1
    (C Head(1 tl l)Emp Emp)
    ([n2][if (Head(Succ m1 tl l)<=Lb u)
       (C Lb u(cIns Head(Succ m1 tl l)m1 L u)R u)
       (C Lb u L u(cIns Head(Succ m1 tl l)m1 R u))])]])
```

Represents the function g of type $\mathbf{L}(\mathbf{N}) \to \mathbf{N} \hookrightarrow \mathbf{N} \hookrightarrow \mathbf{T}$ with

$$g(l,n,0) := \diamond, \qquad g(l,n,m+1) :=$$

$$\begin{cases} u & \text{if } |l| \le m, \\ C_{\mathrm{hd}(\mathrm{tl}_1(l))}(\diamond,\diamond), & \text{if } 0 = m < |l|, \\ C_{\mathrm{Lb}(u)}(f(a,m,L(u)),R(u)) & \text{if } 0 < m < |l| \text{ and } a \le \mathrm{Lb}(u) \\ C_{\mathrm{Lb}(u)}(L(u),f(a,m,R(u))) & \text{if } 0 < m < |l| \text{ and } \mathrm{Lb}(u) < a \end{cases}$$

where $u := g(l,n,m)$ and $a := \mathrm{hd}(\mathrm{tl}_{m+1}(l))$.

Specializing the Treesort Lemma to l, n, n we obtain

$$|l| \leq n \to \exists_u S(l, u).$$

Let $\bar{S}(l, l')$ express that l' is multiset-equal to l and sorted. One easily proves $S(l, u) \to \bar{S}(l, \mathrm{Flatten}(u))$ and gets

$$|l| \leq n \to \exists_{l'} \bar{S}(l, l')$$

in $\mathrm{LA}(;) + \mathrm{Flatten}$. The term extracted from the proof represents the function h of type $\mathbf{L}(\mathbf{N}) \to \mathbf{N} \hookrightarrow \mathbf{L}(\mathbf{N})$ with

$$h(l, n) := \mathrm{Flatten}(g(l, n, n))$$

and thus the treesort algorithm.

Conclusion

- Constructive logic (and arithmetic) can and should be seen as an extension of the classical setup.
- Using the realizability interpretation of proofs one can extract computational content.
- Verification can be automated: there is an internal proof of the soundness theorem.