Linear Types and Non-size-increasing Polynomial Time Computation

Martin Hofmann
LICS 1999

Sukyoung Ryu

ROPAS, KAIST

April 27, 2000

Preliminaries

- Problem

  a type system which ensures that all functions definable in the system are polynomial time computable

- Size of natural number

  the length of its binary representation ($|x| = \lceil \log_2(x+1) \rceil$)

- Ordinary primitive recursion

  which expresses a value $f(x)$ in terms of $f(x-1)$ will almost lead to exponential (in $|x|$) runtime of f

- **Definition 1 (Recursion on notation)**

  Let A be a set, $g \in A$, and $h \in Nat \times A \longrightarrow A$. The function $f : Nat \longrightarrow A$ is defined from g, h by recursion on notation, *if*

$$f(0) = g$$
$$f(x) = h(x, f(\lfloor \tfrac{x}{2} \rfloor)), \quad \text{when } x > 0$$

  The set A is called the result type, *the function h is called the* step function *of the recursive definition*

Previous work[1]

- "*predicativity* and *linearity* suffice to ensure polynomial runtime of all first-order programs"
  - predicativity
    the step function is not allowed to recurse on the result of a previous function call

$$
\begin{aligned}
f(0) &= 1 & \qquad g(0) &= 2 \\
f(x) &= 4(f(\lfloor \tfrac{x}{2} \rfloor)) & \qquad g(x) &= f(g(\lfloor \tfrac{x}{2} \rfloor))
\end{aligned}
$$

  - linearity
    the step function should use its argument at most once

$$
s(x) = 2x \qquad
\begin{aligned}
h(0) &= s \\
h(x) &= h(\lfloor \tfrac{x}{2} \rfloor) \circ h(\lfloor \tfrac{x}{2} \rfloor)
\end{aligned}
$$

- low expressivity

```
insert (a, []) = [a]
insert (a, b::l) = if a <= b then a::b::l
                             else b::insert(a, l)
sort ([]) = []
sort (a::l) = insert(a, sort(l))
```

$\Longrightarrow$ polynomial time but not predicative

[1] [1] S.Bellantoni, K.-H. Niggl, and H. Schwichtenberg. Ramification, Modality, and Linearity in Higher Type Recursion. in preparation, 1998.
[2] Martin Hofmann. Typed lambda calculi for polynomial-time computation, 1998. Habilitation thesis, TU Darmstadt, Germany. To appear as Edinburgh University Technical Report.

This paper

- Observation

  the insertion function does not increase the size of its input

- A linear type system which has the property that all definable functions are *non-size-increasing* without any predicativity restriction

  (*non-size-increasing*: do not increase the size of their input)

Affine Linear Lambda Calculus

$$
\begin{array}{llcll}
\textit{Type} & \tau & ::= & \mathtt{b} & \text{base type} \\
 & & | & \tau_1 \multimap \tau_2 & \text{linear abstraction type} \\
 & & | & \tau_1 \rightarrow \tau_2 & \text{operator type} \\
 & & | & \tau_1 \otimes \tau_2 & \text{tensor product type} \\
 & & | & \tau_1 \times \tau_2 & \text{product type}
\end{array}
$$

$$
\begin{array}{llcll}
\textit{Expr} & e & ::= & op(e_1, \cdots, e_n) & \text{operation} \\
 & & | & x & \text{variable} \\
 & & | & \lambda x : \tau . e & \text{immediate function} \\
 & & | & e_1 \; e_2 & \text{application} \\
 & & | & e_1 \otimes e_2 & \text{tensor product} \\
 & & | & \mathtt{let}\ e_1 \mathtt{=} x \otimes y\ \mathtt{in}\ e_2 & \text{let binding} \\
 & & | & (e_1,\ e_2) & \text{product} \\
 & & | & e.1 & \text{projection} \\
 & & | & e.2 & \text{projection}
\end{array}
$$

Static Semantics

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \; \texttt{(T-VAR)}$$

$$\frac{\Gamma, x{:}\,\tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau . e : \tau_1 \multimap \tau_2} \; \texttt{(T-ARR-I)}$$

$$\frac{\Gamma_1 \vdash e_1 : \tau_1 \multimap \tau_2 \qquad \Gamma_2 \vdash e_2 : \tau_1}{\Gamma_1, \Gamma_2 \vdash e_1 \; e_2 : \tau_2} \; \texttt{(T-ARR-E)}$$

$$\frac{\Gamma_1 \vdash e_1 : \tau_1 \qquad \Gamma_2 \vdash e_2 : \tau_2}{\Gamma_1, \Gamma_2 \vdash e_1 \otimes e_2 : \tau_1 \otimes \tau_2} \; \texttt{(T-TENS-I)}$$

$$\frac{\Gamma_1 \vdash e_1 : \tau_1 \otimes \tau_2 \qquad \Gamma_2, x : \tau_1, y : \tau_2 \vdash e_2 : \tau_3}{\Gamma_1, \Gamma_2 \vdash \texttt{let } e_1 \texttt{ = } x \otimes y \texttt{ in } e_2 : \tau_3} \; \texttt{(T-TENS-E)}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1,\ e_2) : \tau_1 \times \tau_2} \; \texttt{(T-PROD-I)}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \times \tau_2 \qquad i \in \{1, 2\}}{\Gamma \vdash e.i : \tau_i} \; \texttt{(T-PROD-E)}$$

$$\frac{\emptyset \vdash op : \tau_1 \times \cdots \times \tau_n \to \tau \qquad \emptyset \vdash e_i : \tau_i \qquad \text{for } i = 1 \cdots n}{\emptyset \vdash op(e_1, \cdots, e_n) : \tau} \; \texttt{(T-OP)}$$

$\Gamma \vdash e : \tau$ – all the free variables of e are mentioned in Γ
and they are *used* at most once in e

Non-size-increasing Property

- Set-theoretic interpretation $[\![]\!] : \mathit{Type} \to 2^{\mathit{Val}}$

$$
\begin{array}{lcl}
[\![\mathtt{N}]\!] & = & \mathit{Nat} \\
[\![\tau_1 \multimap \tau_2]\!] & = & \{f \mid v \in [\![\tau_1]\!], \quad f(v) \in [\![\tau_2]\!]\} \\
[\![\tau_1 \otimes \tau_2]\!] & = & \{v_1 \otimes v_2 \mid v_1 \in [\![\tau_1]\!], \quad v_2 \in [\![\tau_2]\!]\} \\
[\![\tau_1 \times \tau_2]\!] & = & \{(v_1, v_2) \mid v_1 \in [\![\tau_1]\!], \quad v_2 \in [\![\tau_2]\!]\}
\end{array}
$$

- Size function $\mathtt{s}_\tau : [\![\tau]\!] \to \mathit{Nat}$

$$
\begin{array}{lcl}
\mathtt{s}_{\mathtt{N}}(x) & = & |x| \\
\mathtt{s}_{\tau_1 \otimes \tau_2}(u \otimes v) & = & \mathtt{s}_{\tau_1}(u) + \mathtt{s}_{\tau_2}(v) \\
\mathtt{s}_{\tau_1 \times \tau_2}((u, v)) & = & \max(\mathtt{s}_{\tau_1}(u), \mathtt{s}_{\tau_2}(v)) \\
\mathtt{s}_{\tau_1 \multimap \tau_2}(f) & = & \min\{c \mid \forall v \in [\![\tau_1]\!].\mathtt{s}_{\tau_2}(f(v)) \leq c + \mathtt{s}_{\tau_1}(v)\}
\end{array}
$$

- **Proposition 2 (Non-size-increasing property)**

  Suppose that for each op whose type is $\tau_1 \times \cdots \times \tau_n \to \tau$, $v_i \in [\![\tau_i]\!]$, *and* $\mathtt{s}_{\tau_i}(v_i) = 0$, *we have* $\mathtt{s}_\tau(op(v_1, \cdots, v_n)) = 0$.
  If $\Gamma \vdash e : \tau$ *and* $\eta \models e \xrightarrow{*} v$ *then* $\mathtt{s}_\tau(v) \leq \Sigma_{x \in dom(\Gamma)} \mathtt{s}_{\Gamma(x)}(\eta(x))$.

Examples - Natural Numbers

- a constant

$$\mathtt{0} : \mathtt{N} \qquad\qquad \mathtt{s}_{\mathtt{N}}(\mathtt{0}) = 0$$

- binary successor functions

$$\begin{array}{lll} \mathtt{suc}_{\mathtt{0}} : \mathtt{N} \multimap \mathtt{N} & \mathtt{suc}_{\mathtt{0}}(x) = 2x & \mathtt{s}_{\mathtt{N} \multimap \mathtt{N}}(\mathtt{suc}_{\mathtt{0}}) = 1 \\ \mathtt{suc}_{\mathtt{1}} : \mathtt{N} \multimap \mathtt{N} & \mathtt{suc}_{\mathtt{1}}(x) = 2x+1 & \mathtt{s}_{\mathtt{N} \multimap \mathtt{N}}(\mathtt{suc}_{\mathtt{1}}) = 1 \end{array}$$

$$\begin{array}{rcl} \mathtt{s}_{\mathtt{N} \multimap \mathtt{N}}(\mathtt{suc}_{\mathtt{0}}) & = & \min\{c \mid \forall v \in [\![\mathtt{N}]\!].\mathtt{s}_{\mathtt{N}}(\mathtt{suc}_{\mathtt{0}}(v)) \leq c + \mathtt{s}_{\mathtt{N}}(v)\} \\ & = & \min\{c \mid \forall v \in [\![\mathtt{N}]\!].\mathtt{s}_{\mathtt{N}}(2v) \leq c + \mathtt{s}_{\mathtt{N}}(v)\} \\ & = & \min\{c \mid \forall v \in [\![\mathtt{N}]\!].|2v| \leq c + |v|\} \\ & = & \min\{c \mid \forall v \in [\![\mathtt{N}]\!].\lceil \log_2 2v \rceil \leq c + \lceil \log_2 v \rceil\} \\ & = & \min\{c \mid \forall v \in [\![\mathtt{N}]\!].1 + \lceil \log_2 v \rceil \leq c + \lceil \log_2 v \rceil\} \\ & = & 1 \end{array}$$

- a new base type $\diamond$

$$\diamond : \Diamond \qquad\qquad \mathtt{s}_{\Diamond}(\diamond) = 1$$

- new binary successor functions

$$\begin{array}{lll} \mathtt{suc}_{\mathtt{0}} : \Diamond \multimap \mathtt{N} \multimap \mathtt{N} & \mathtt{suc}_{\mathtt{0}}(\diamond)(x) = 2x & \mathtt{s}_{\Diamond \multimap \mathtt{N} \multimap \mathtt{N}}(\mathtt{suc}_{\mathtt{0}}) = 0 \\ \mathtt{suc}_{\mathtt{1}} : \Diamond \multimap \mathtt{N} \multimap \mathtt{N} & \mathtt{suc}_{\mathtt{1}}(\diamond)(x) = 2x & \mathtt{s}_{\Diamond \multimap \mathtt{N} \multimap \mathtt{N}}(\mathtt{suc}_{\mathtt{1}}) = 0 \end{array}$$

$$\begin{array}{rcl} \mathtt{s}_{\Diamond \multimap \mathtt{N} \multimap \mathtt{N}}(\mathtt{suc}_{\mathtt{0}}) & = & \min\{c \mid \mathtt{s}_{\mathtt{N} \multimap \mathtt{N}}(\mathtt{suc}_{\mathtt{0}}(\diamond)) \leq c + \mathtt{s}_{\Diamond}(\diamond)\} \\ & = & \min\{c \mid 1 \leq c + 1\} \\ & = & 0 \end{array}$$

Expressivity and Soundness

- Expressivity
  - can not characterise the functions definable in affine linear lambda calculus with non-size-increasing property yet
  - all functions computable in polynomial time are representable

- **Theorem 3 (Soundness)**

  If $\emptyset \vdash e : \mathtt{N} \multimap \mathtt{N}$ *then* e *is polynomial time computable.*