

Informal meeting on the article

Type Systems for Polynomial-time Computations

(Habilitation thesis of Martin Hofmann)

Yu Zhang

March 23, 2007



Sophia-Antipolis — March 23, 2007

Background

- ❖ Programming languages characterising complexity classes.
 - Applications with resource restrictions.
 - Verification of cryptographic algorithms.
- ❖ Recursion schemes capturing PTIME computations:
 - Bounded recursion [Cobham 1963, ...]
 - Safe recursion [Bellantoni & Cook]
 - Tiered recursion [Leivant & Marion]
- ❖ Surveys:
 - Programming languages capturing complexity classes*, by M. Hofmann.
 - Computation models and function algebras*, by P. Clote.

Contents of the paper

- ❖ A typed λ -calculus **SLR** with
 - a mixed use of linearity and modality, and
 - a decidable type checking system.
- ❖ A category-theoretic proof of soundness:
 - a categorical model of SLR;
 - all morphisms are PTIME computable functions by construction.

Recursion on notation

- ❖ Complexity is measured in terms of the length of binary representations: $|x| = \lceil \log_2(x + 1) \rceil$.
- ❖ Primitive recursion usually makes an exponential (of $|x|$) number of recursive calls.
- ❖ Recursion on notation:

$$\begin{aligned}f(0, \vec{y}) &= g(\vec{y}) \\f(x, \vec{y}) &= h(x, \vec{y}, f(\lfloor \frac{x}{2} \rfloor, \vec{y})), \text{ when } x > 0\end{aligned}$$

An evaluation of $f(x, \vec{y})$ requires $|x|$ recursive calls.

- ❖ Alternative syntax: a simply-typed λ -calculus.

$$\begin{aligned}S_0, S_1 &: \mathbb{N} \rightarrow \mathbb{N} \\ \text{rec} &: \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N} \rightarrow \mathbb{N}\end{aligned}$$

An example: recursion on notation is NOT enough!

$$sq(0) = 1$$

$$sq(x) = 4 \cdot sq(\lfloor \frac{x}{2} \rfloor), \text{ when } x > 0$$

$sq(x) = (2^{|x|})^2$ is a function of polynomial growth, but

$$exp(0) = 2$$

$$exp(x) = sq(exp(\lfloor \frac{x}{2} \rfloor)), \text{ when } x > 0$$

$exp(x) = 2^{(3 \cdot 2^{|x|} - 2)}$ is a function of **exponential** growth.

An example: recursion on notation is NOT enough!

$$sq(0) = 1$$

$$sq(x) = 4 \cdot sq(\lfloor \frac{x}{2} \rfloor), \text{ when } x > 0$$

$sq(x) = (2^{|x|})^2$ is a function of polynomial growth, but

$$exp(0) = 2$$

$$exp(x) = sq(exp(\lfloor \frac{x}{2} \rfloor)), \text{ when } x > 0$$

$exp(x) = 2^{(3 \cdot 2^{|x|} - 2)}$ is a function of **exponential** growth.

- Results of recursive calls should not be used as arguments of recursively defined functions.

Safe recursion [Bellantoni & Cook]

- ❖ Variables are divided into **normal** and **safe** variables: $f(\vec{x}; \vec{y})$.
- ❖ Safe composition: everything plugged in a normal position is defined with only normal variables:

$$k(\vec{x}; \vec{y}) = f(\vec{g}(\vec{x};); \vec{h}(\vec{x}; \vec{y}))$$

- ❖ Results of recursive calls are used via safe variables only:

$$\begin{aligned} f(0, \vec{y}; \vec{z}) &= g(\vec{y}; \vec{z}) \\ f(x, \vec{y}; \vec{z}) &= h(x, \vec{y}; f(\lfloor \frac{x}{2} \rfloor, \vec{y}; \vec{z}), \vec{z}) \end{aligned}$$

The function exp is not definable by safe recursion:

$$exp(x;) = sq(exp(\lfloor \frac{x}{2} \rfloor;);)$$

Alternative syntax: modal types

Types from the modal logic:

➤ a unary type constructor \Box :

$$\frac{\Gamma \vdash e : \Box(\tau)}{\Gamma \vdash e : \tau} \quad \frac{\Gamma \vdash e : \tau \quad \Gamma(x) = \Box(_) \text{ for all } x \in FV(e)}{\Gamma \vdash e : \Box(\tau)}$$

➤ $\Box(\tau)$ are types for normal variables.

➤ the safe recursor:

$$\text{rec} : \mathbb{N} \rightarrow (\Box(\mathbb{N}) \rightarrow \mathbb{N} \rightarrow \mathbb{N}) \rightarrow \Box(\mathbb{N}) \rightarrow \mathbb{N}$$

The function *exp* is not typable:

$$\begin{aligned} sq &\equiv \text{rec}(\mathbf{S}_1\mathbf{0})(\lambda y^{\Box(\mathbb{N})}. \lambda z^{\mathbb{N}}. \mathbf{S}_0(\mathbf{S}_0 z)) : \Box(\mathbb{N}) \rightarrow \mathbb{N} \\ exp &\equiv \text{rec}(\mathbf{S}_0(\mathbf{S}_1\mathbf{0}))(\lambda y^{\Box(\mathbb{N})}. \lambda z^{\mathbb{N}}. sq(z)) : \mathbf{X} \end{aligned}$$

Higher-type recursion and linearity

- Higher-type recursor

$$\mathbf{rec}^\tau : \tau \rightarrow (\square(\mathbf{N}) \rightarrow \tau \rightarrow \tau) \rightarrow \square(\mathbf{N}) \rightarrow \tau$$

- This recursor breaks the PTIME restriction:

$$f \equiv \mathbf{rec}^{\mathbf{N} \rightarrow \mathbf{N}} \mathbf{S}_0(\lambda n^{\square(\mathbf{N})}. \lambda g^{\mathbf{N} \rightarrow \mathbf{N}}. \lambda z^{\mathbf{N}}. g(g(z))) : \square(\mathbf{N}) \rightarrow \mathbf{N} \rightarrow \mathbf{N}$$

$$f(x, y) = 2^{2^{|x|}} \cdot y \text{ is of exponential growth.}$$

- Results of recursive calls should be used **linearly**.
- Linear functions: arguments can be used at most once.

$$\mathbf{rec}^\tau : \tau \multimap (\square(\mathbf{N}) \multimap \tau \multimap \tau) \multimap \square(\mathbf{N}) \multimap \tau$$

SLR system – types, terms

❖ Types:

τ, τ', \dots	::=	\mathbf{N}	natural numbers
		$\tau \multimap \tau'$	linear functions
		$\tau \rightarrow \tau'$	nonlinear, nonmodal (optional)
		$\Box \tau \rightarrow \tau'$	modal functions
		\dots	

\Box itself is NOT a type constructor in SLR.

❖ Terms:

$$e, e', \dots ::= \lambda x. e \mid ee' \mid \mathbf{0} \mid \mathbf{S}_0 \mid \mathbf{S}_1 \mid \mathbf{rec}^\tau \mid \mathbf{case}^\tau \mid \dots$$

Above is a non-polymorphic fragment of the SLR system.

SLR system – aspects, subtyping, contexts

❖ Aspects:

(nonlinear, modal) < (nonlinear, nonmodal) < (linear, nonmodal)

❖ Subtyping:

$$\frac{\tau' <: \tau \quad \sigma <: \sigma' \quad a' \leq a}{\tau \xrightarrow{a} \sigma <: \tau' \xrightarrow{a'} \sigma'}$$

In particular, $\tau \multimap \tau' <: \tau \rightarrow \tau' <: \Box \tau \rightarrow \tau'$.

❖ Typing contexts assign to variables **aspects** as well as types:

$$x_1 :^{a_1} \tau_1, \dots, x_n :^{a_n} \tau_n$$

Aspects specify the way how variables can be used in terms.

SLR system – typing rules

$$\frac{\Gamma \vdash e : \tau \quad \tau <: \tau'}{\Gamma \vdash e : \tau'}$$

$$\frac{\Gamma, x :^a \tau \vdash e : \tau'}{\Gamma \vdash \lambda x. e : \tau \xrightarrow{a} \tau'}$$

$$\frac{\Gamma, \Delta_1 \vdash e_1 : \tau \xrightarrow{a} \tau' \quad \Gamma, \Delta_2 \vdash e_2 : \tau \quad \Gamma \text{ nonlinear} \quad a' \leq a \text{ for all } x :^{a'} \in \Gamma, \Delta_2}{\Gamma, \Delta_1, \Delta_2 \vdash e_1 e_2 : \tau'}$$

$$\frac{}{\Gamma \vdash \text{rec}_\tau : \tau \multimap \square(\square N \rightarrow \tau \multimap \tau) \rightarrow \square N \rightarrow \tau}$$

Example:

$f \equiv \text{rec}^{N \multimap N} S_0 h : \square N \rightarrow N \multimap N$ where we are expecting

$$\vdash h \equiv \lambda n^N. \lambda g^{N \multimap N}. \lambda z^N. g(g(z)) : \square N \rightarrow (N \multimap N) \multimap N \multimap N$$

but in SLR we can only infer

$$\vdash h : \square N \rightarrow (N \multimap N) \rightarrow N \multimap N \quad \text{or} \quad \vdash h : \square N \rightarrow \square(N \multimap N) \rightarrow N \multimap N$$

Main result

- ❖ Type-checking in SLR is decidable.
- ❖ All definable numeric functions are PTIME computable.
 - A category-theoretic model for SLR.
 - All morphisms are PTIME computable by construction.

Main result

- ❖ Type-checking in SLR is decidable.
- ❖ All definable numeric functions are PTIME computable.
 - A category-theoretic model for SLR.
 - All morphisms are PTIME computable by construction.
- ❖ [Murawski & Ong] SLR- is PTIME complete.
 - SLR- is SLR with only two function spaces: $_ \multimap _$ and $\square _ \rightarrow _$.
 - The proof is standard: encoding PTIME Turing machines in SLR-.