Typed $\lambda$-calculi with explicit substitutions may not terminate

Paul-André Mellies *

Ecole Normale Supérieure, 45 rue d’Ulm, 75005 Paris, France
INRIA Rocquencourt, Domaine de Voluceau, 78153 Le Chesnay Cedex, France
FWI, De Boelelaan 1081a, 1081 HV Amsterdam, Nederland
mellies@cs.vu.nl

Abstract. We present a simply typed $\lambda$-term whose computation in the $\lambda\sigma$-calculus does not always terminate.

1 The $\lambda\sigma$-calculus, introduction

Any effective implementation of the $\lambda$-calculus requires some control on the substitution to benefit from graph sharing [1] and avoid immediate size explosion. The original $\lambda$-calculus cannot describe these controls an easy way. The $\lambda\sigma$-calculus was introduced in [2] as a bridge between the classical $\lambda$-calculus and its concrete implementations. Substitutions become explicit, they can be delayed and stored. The calculus provides a pleasant setting to study substitutions and check implementations.

The syntax of the $\lambda\sigma$-calculus contains two classes of objects: terms and substitutions. Terms are written in the De Bruijn notation [3].

Terms

$$a ::= \permuted 1|ab|\lambda a|a[s]$$

Substitutions

$$s ::= \text{id}||a\cdot s|s\circ t$$

The rule $\text{Beta}$ is equivalent to the usual $\beta$-rule of the $\lambda$-calculus. The other rules, called $\sigma$-rules, expose how substitutions are pushed inside the terms and performed.

$$\text{Beta} \quad (\lambda a)b \rightarrow a[b \cdot \text{id}]$$

$$\text{App} \quad (ab)[s] \rightarrow a[s]b[s]$$

$$\text{Abs} \quad (\lambda a)[s] \rightarrow \lambda(a[1.(s \circ t)])$$

$$\text{Clos} \quad a[s][t] \rightarrow a[s \circ t]$$

$$\text{Map} \quad (a \cdot s) o t \rightarrow a[t] \cdot (s o t)$$

$$\text{Ass} \quad (s_1 \circ s_2) \circ s_3 \rightarrow s_1 \circ (s_2 \circ s_3)$$

$$\text{VarId} \quad 1[id] \rightarrow 1$$

$$\text{VarCons} \quad 1[a.s] \rightarrow a$$

$$\text{IdL} \quad \text{id} \circ s \rightarrow s$$

$$\text{ShiftId} \quad \uparrow \circ \text{id} \rightarrow \uparrow$$

$$\text{ShiftCons} \quad \uparrow \circ (a \cdot s) \rightarrow s$$

* This work was partly supported by the Esprit BRA CONFER.
When carried out inside the $\lambda$-calculus, any reduction of a typed $\lambda$-term $M$ reaches its normal form. Some $\lambda\sigma$-reductions can mimic the $\lambda$-reductions and terminate too. Others can be more subtle and compute $M$ in a non-standard way. However, does any $\lambda\sigma$-computation of a typed term normalise it? The question was much debated and investigated with hopes for a positive answer. The major clue was the strong normalisation of the $\sigma$-rules which was proved effective in [4] and then [5][6] on any $\lambda\sigma$-term. It makes a non terminating $\lambda\sigma$-computation continually create and reduce new Beta-redexes, which seems to contradict the typed structure of the term.

However, we present here a closed and simply typed $\lambda$-term whose computation in the $\lambda\sigma$-calculus does not always terminate. The $\lambda\sigma$-reductions are thus not strictly bound to the $\lambda$-reductions, which is a surprise.

2 Basic intuitions

Let $M$ be the simply typed $\lambda$-term $\lambda v. (\lambda x. (\lambda y. y)((\lambda z. z)x)((\lambda w. w)v))$. Like any typed term its $\lambda\sigma$-computation may normalise it. Next section, we show that it may also not terminate.

Building such a non terminating strategy on $M$ requires precision. The $\sigma$-rules enjoy strong normalisation on any $\lambda\sigma$-term. The Beta-rule mimics the $\beta$-rule whose computation on any well typed $\lambda$-term strongly terminates. This shows that non termination must come from thin interactions between the Beta and $\sigma$-rules. Let $(\lambda a)b$ be a $\lambda$-term and $s$ a substitution on top of it. We study next two natural strategies to reduce the root Beta-redex and begin the propagation of $s$.

One standard strategy begins to reduce the Beta-redex

$$((\lambda a)b)[s] \rightarrow (a[b \cdot id])[s]$$

and then propagate the two substitutions $s$ and $(b \cdot id)$ inside $a$ using $\sigma$-rules. If carried on, the $\sigma$-computation terminates on a $\lambda$-term $c$.

Another natural strategy begins with the two $\sigma$-rules App and Lambda in order to propagate $s$ through the Beta-redex. We call $s$ and $s'$ the two copies of $s$ by App.

$$((\lambda a)b)[s] \rightarrow ((\lambda a)[s])b[s']$$

App

$$\rightarrow (\lambda(a[s \circ T]))b[s']$$

Lambda

It then computes the root Beta-redex:

$$\rightarrow a[1 \cdot s \circ T][b[s'] \cdot id]$$

Beta

The two substitutions $(1 \cdot (s \circ T))$ and $(b[s'] \cdot id)$ are then propagated inside $a$ using $\sigma$-rules. If carried on the process terminates again on the same $\lambda$-term $c$. 