An Approach to Call-by-Name Delimited Continuations

(version with errata - mid-Nov 2007)

Hugo Herbelin
INRIA Futurs, France
Hugo.Herbelin@inria.fr

Silvia Ghilezan
Faculty of Engineering, University of Novi Sad, Serbia
gsilvia@uns.ns.ac.yu

Abstract

We show that a variant of Parigot's $\lambda\mu$-calculus, originally due to de Groote and proved to satisfy Böhm's theorem by Saurin, is canonically interpretable as a call-by-name calculus of delimited control. This observation is expressed using Ariola et al's call-by-value calculus of delimited control, an extension of $\lambda\mu$-calculus with delimited control known to be equationally equivalent to Danvy and Filinski's calculus with *shift* and *reset*. Our main result then is that de Groote and Saurin's variant of $\lambda\mu$-calculus is equivalent to a canonical call-by-name variant of Ariola et al's calculus. The rest of the paper is devoted to a comparative study of the call-by-name and call-by-value variants of Ariola et al's calculus, covering in particular the questions of simple typing, operational semantics, and continuation-passing-style semantics. Finally, we discuss the relevance of Ariola et al's calculus as a uniform framework for representing different calculi of delimited continuations, including "lazy" variants such as Sabry's *shift* and *lazy reset* calculus.

Categories and Subject Descriptors F.3.3 [*Studies of Program Constructs*]: Control primitives; F.4.1 [*Mathematical Logic*]: Lambda calculus and related systems

Keywords Delimited control, Observational completeness, Böhm separability, Classical logic.

1. Introduction

Control calculi emerged as an attempt to abstractly characterise the semantics of operators like Scheme's `call/cc` that capture the current continuation of a computation. One first such calculus is the $\lambda_{\mathcal{C}}$-calculus of Felleisen et al. (1986). Control operator are connected to classical logic, as first investigated by Griffin (1990). Hence, it is not a surprise that the "cleanest" such λ-calculus of control, namely $\lambda\mu$-calculus of Parigot (1992) comes from a computational analysis of classical natural deduction: as shown by Ariola and Herbelin (2007), $\lambda\mu$-calculus extended with a single "toplevel" continuation constant `tp` provides a fine-grained calculus able, among other things, to faithfully express the operational semantics of `call/cc`, $\mathcal{C}$, etc, including its own operational semantics, a property that $\lambda_{\mathcal{C}}$-calculus achieves only at observational level. The reason for this success is that $\lambda\mu$-calculus treats evaluation contexts as stand-alone first-class objects while $\lambda_{\mathcal{C}}$ manages evaluation contexts through their reification as regular functions.

POPL'08, January 7–12, 2008, San Francisco, California, USA.

Delimited control and completeness properties

If we concentrate on call-by-value control calculi, the introduction of delimiters can be traced back to Johnson (1987), Johnson and Duggan (1988), Felleisen (1988), and Danvy and Filinski (1989). It has then been shown in different contexts that adding such delimiters increases the expressiveness of control calculi. For instance, Sitaram and Felleisen (1990) showed how to recover a full abstraction result for call-by-value PCF with control by adding a control delimiter. As another striking example, Filinski (1994) showed that delimited control is complete for representing concrete monads, hence to simulate side-effects such as states, exceptions, etc.

Historically, delimited control came with ad hoc operators for composing continuations: Felleisen had a calculus that included a delimiter *prompt* and a control operator *control* (also respectively written $\#$ and $\mathcal{F}$); Danvy and Filinski had an operator *shift* to compose continuations and an operator *reset* to delimit them (these were also written $\mathcal{S}$ and $\langle\_\rangle$). From Filinski (1994), it is known that *shift* and *reset* are equivalent to the combination of Scheme's `call/cc`, Felleisen's *abort* and *reset*, and hence equivalent to $\mathcal{C}$ and *reset*. From Shan (2004), it is known that *control* and *prompt* are also equivalent to *shift* and *reset*, in spite that *control* is semantically more complex to study than $\mathcal{C}$ or *shift*. The simplicity of the semantics of *shift* together with its relevance for some programming applications contributed to set *shift* as a reference in delimited control. And this is so in spite (it seems that) it has never been studied until now as part of a dedicated λ-calculus of delimited control.

As shown by Ariola et al. (2007), a fine-grained calculus of delimited control of the strength of *shift* and *reset* is obtained if one starts from $\lambda\mu$-calculus and extends it first by a notation `tp` for the "toplevel" continuation, then by a toplevel delimiter. A possible interpretation for this toplevel delimiter is as a *dynamic* binder of `tp`, what justifies to interpret the resulting call-by-value calculus, called $\lambda\mu\widehat{\texttt{tp}}$, as an extension of call-by-value $\lambda\mu$-calculus with a single dynamically bound continuation variable $\widehat{\texttt{tp}}$, where the hat on `tp` emphasises the dynamic treatment of the variable. A typical analogy for the dynamic continuation variable here is exception handling: each call to $\widehat{\texttt{tp}}$ is dynamically bound to the closest surrounding $\widehat{\texttt{tp}}$ binder, in exactly the same way as a raised exception is dynamically bound to the closest surrounding handler.

On the call-by-name side, we know no explicit mention of delimited continuations, but two results related to Böhm's theorem (a form of observational completeness stated as a separability property) raised interesting questions: David and Py (2001) showed that Parigot's $\lambda\mu$-calculus *does not* satisfy Böhm's theorem while Saurin (2005) showed that an apparently inoffensive variant of $\lambda\mu$-calculus due to de Groote (1994) *does* satisfy Böhm's theorem.

Until Saurin's result, de Groote's variant of Parigot's $\lambda\mu$-calculus was merely considered in typed settings, and more particularly in settings where the continuation calls had type $\bot$ (de Groote 1994; Ong 1996; Ong and Stewart 1997; Selinger 2001, non exaustive list). But in such a typed setting, de Groote's calculus is not far from being equivalent[1] to Parigot's $\lambda\mu$-calculus extended with a single continuation constant of type $\bot$. Hence, Saurin's result is the first result revealing that de Groote's calculus is strictly stronger than Parigot's one in the untyped setting. In our opinion, this justifies to refer to this calculus as de-Groote–Saurin's calculus. Our main result then is that de-Groote–Saurin's calculus can be interpreted as a canonical call-by-name variant of call-by-value $\lambda\mu\widehat{\mathtt{tp}}$.

Capitalising on the equational correspondence between call-by-value $\lambda\mu\widehat{\mathtt{tp}}$ and an axiomatic of Danvy and Filinski's *shift* and *reset* given by (Kameyama and Hasegawa 2003), we can then assert that the calculus with *shift* and *reset* and de-Groote–Saurin's calculus are two facets of the very same notion of delimited control.

Outline of the paper

Section 2 is a brief survey of Parigot's $\lambda\mu$ and de Groote's variant of $\lambda\mu$, including the separability properties studied by David and Py, and by Saurin. Section 3 presents call-by-value $\lambda\mu\widehat{\mathtt{tp}}$ and its relation with *shift* and *reset*. It reviews the results by Ariola et al. (2007) and completes them by a formal presentation of the operational semantics of call-by-value $\lambda\mu\widehat{\mathtt{tp}}$. In Section 4 we introduce a call-by-name $\lambda\mu\widehat{\mathtt{tp}}$ and show that it is equivalent to de-Groote–Saurin's calculus. Especially, it directly inherits separability from it. We study call-by-name $\lambda\mu\widehat{\mathtt{tp}}$ in comparison with the call-by-value $\lambda\mu\widehat{\mathtt{tp}}$: we propose a system of simple types for which subject reduction holds and we study the operational and continuation-passing-style semantics. A further analysis of $\lambda\mu\widehat{\mathtt{tp}}$ leads to a classification of four calculi of delimited continuations which is discussed in Section 5. Concluding remarks are given in Section 6.

2. Parigot's $\lambda\mu$-Calculus and Saurin's $\Lambda\mu$-Calculus

Failure of separability in $\lambda\mu$-calculus The $\lambda\mu$-calculus (Parigot 1992), for short $\lambda\mu$, is an untyped calculus designed to computationally interpret proofs of classical natural deduction. Its syntax is defined by the following grammar:

Parigot's $\lambda\mu$-calculus

$$\begin{array}{lcll} M & ::= & x \mid \lambda x.M \mid M\,M \mid \mu\alpha.c & \text{(terms)} \\ c & ::= & [\alpha]M & \text{(commands)} \end{array}$$

where x, y, z and their notational derivatives range over an infinite set of *term variables* and α, β, γ, δ and their notational derivatives range over an infinite set of *continuation variables* (also called *evaluation context variables*). *Expressions* contain *terms* (called *unnamed terms* in Parigot) and *commands* (called *named terms* in Parigot). The operators λ and μ are binders. Free and bound variables are defined as usual and we reason modulo renaming of bound variables. A term or command is *closed* if it contains no free variables.

The calculus is equipped with the call-by-name reduction rules shown in Figure 1. The notations $M[N/x]$ and $c[\beta/\alpha]$ denote usual capture-free substitutions, whereas the expression $c[[\beta](\Box\ N)/\alpha]$,

[1] If we call $\mathtt{tp}_\bot$ the continuation constant of type $\bot$, then it is enough to enrich the calculus of de Groote (1994) with the rule (ϵ) of de Groote (1998) to get, through the interpretation of de Groote's $\mu\alpha.t$ and $[\alpha]t$ as $\mu\alpha.[\mathtt{tp}_\bot]t$ and $\mu\delta.[\alpha]t$ (δ fresh) respectively, a calculus equivalent to Parigot's calculus.

$$\begin{array}{llcll} \beta: & (\lambda x.M)\,N & \to & M[N/x] & \\ \mu_{app}: & (\mu\alpha.c)\,N & \to & \mu\beta.c[[\beta](\Box\ N)/\alpha] & \beta \text{ fresh} \\ \mu_{var}: & [\beta]\mu\alpha.c & \to & c[\beta/\alpha] & \end{array}$$

Figure 1. Reduction rules of $\lambda\mu$-calculus

$$\begin{array}{llcll} \beta: & (\lambda x.M)\,N & = & M[N/x] & \\ \mu_n: & [\beta](E_n[\mu\alpha.c]) & = & c[[\beta]E_n/\alpha] & \\ \eta_\mu: & \mu\alpha.[\alpha]M & = & M & \text{if } \alpha \text{ not free in } M \\ \eta: & \lambda x.(M\,x) & = & M & \text{if } x \text{ not free in } M \end{array}$$

Figure 2. Equational theory of $\lambda\mu$-calculus

called *structural substitution*, denotes the capture-free substitution of every subterm of the form $[\alpha]M$ in c by $[\beta](M\ \ N)$. For instance, we have $([\alpha](x\ \ \lambda y.\mu\beta.[\alpha]y))[[\gamma](\Box\ \ N)/\alpha] \equiv [\gamma](x\ \lambda y.\mu\beta'.[\gamma](y\ N)\ N)$ where the binder β has been renamed to avoid the capture of possible free occurrences of β in N. Note that the substitution $c[q/\alpha]$ coincides with the structural substitution $c[[q]\Box/\alpha]$. A term that contains no redex is called *normal*. We know from Baba et al. (2001) that call-by-name reduction system is confluent.

The syntax of call-by-name evaluation contexts is defined by $E_n\ ::=\ \Box \mid E_n[\Box\ M]$ where $\Box$ denotes the *hole* of the context. We write $E[M]$ for the term obtained by substituting M in the hole of E and structural substitution extends straightforwardly into a substitution of the form $c[[\beta]E/\alpha]$. We equip $\lambda\mu$ with the equational theory given in Figure 2. Up to the use of (η_μ), the rule (μ_n) is equivalent to the combination of (μ_{var}) and (μ_{app}) so that the equational theory is correctly an extension with η-rules of the reflexive-symmetric-transitive closure of $\to$.

David and Py investigated Böhm's separability for the equational theory of $\lambda\mu$ and showed that it does not satisfy Böhm's separability[2].

PROPOSITION 1 (David and Py 2001). *A normal term is called* canonical *if it contains no subterms of the form $[\alpha]\lambda x.M$. There are two closed canonical normal terms W_0 and W_1 that are not equated by the equational theory in Figure 2 and of which the observational behaviour is not separable, i.e., for distinct fresh variables x and y, there is no applicative context E_n, such that $E_n[W_0] = x$ and $E_n[W_1] = y$.*

Separability in $\Lambda\mu$-calculus Saurin (2005) showed that completeness can be recovered by relaxing the syntax of $\lambda\mu$ so that the category of commands (i.e. named terms) becomes a subcategory of the one of terms. The syntax used by Saurin was already considered by de Groote (1994), Ong (1996), Selinger (2001). This syntax was considered as an alternative to Parigot's original $\lambda\mu$. Saurin's result sheds new light on the relation between the two calculi. Following Saurin, we call $\Lambda\mu$ the calculus based on de Groote's syntax equipped with the same reduction rules and equational theory as in $\lambda\mu$. The syntax is[3]:

$\Lambda\mu$-calculus

[2] David and Py actually had (μ_{var}) and (μ_{app}) instead of (μ_n) and their rules were oriented as rewrite rules. They also considered the rule $\nu:\ \ \mu\alpha.c\ \ \to\ \ \lambda x.\mu\alpha.c[[\alpha](\Box x)/\alpha]$ but this rule is redundant for equational reasoning as it derives from (μ_n) and (η). The initial motivation for (ν) was to turn their system of reduction rules (β), (η), (μ_{app}), (μ_{var}) and (η_μ) into a confluent system of reduction. In fact, (ν) hides an η-expansion and it is enough to formulate (η) in the expansion way to get a confluent system, without any need for (ν).

[3] Saurin's syntax is a bit different as he writes $M\ \alpha$ for what we write $[\alpha]M$ but that is really here a matter of notation.

$$M \quad ::= \quad x \mid \lambda x.M \mid M\,M \mid \mu\alpha.M \mid [\alpha]M$$

In $\Lambda\mu$, there are more evaluation contexts. They are defined by: $D_n \ ::= \ \Box \mid D_n[[\alpha]E_n]$.

THEOREM 2 (Saurin 2005). *A normal term is called* canonical *if it contains no subterms of the form $[\alpha]\lambda x.M$. Now, if the closed canonical normal terms M and N in $\Lambda\mu$ are not equated by the equational theory in Figure 2, then they are separable, i.e., for any two variables x and y, there exists a context D_n, such that $D_n[M] = x$ and $D_n[N] = y$.*

This may look strange as the only change is a change in the syntax of the terms. In fact, the difference lies in the rule (μ_{var}) which in the case of $\lambda\mu$ can only occur in a configuration of the form:

$$M(\mu\gamma.[\beta]\mu\alpha.c) \quad\rightarrow\quad M(\mu\gamma.c[\beta/\alpha])$$

while in the case of $\Lambda\mu$, it can also occur in a configuration of the form:

$$M\,([\beta]\mu\alpha.c) \quad\rightarrow\quad M\,(c[\beta/\alpha])$$

so that the computational effect of any $\mu\alpha.c$ can be cancelled if we succeed in putting it in a context of the form $[\beta]\Box$. This last property is the main reason why Saurin's completeness theorem works.

3. A Review of Call-by-Value $\lambda\mu\widehat{\mathsf{tp}}$-Calculus

The $\lambda_{\mathcal{C}\widehat{\mathsf{tp}}}$-calculus has been introduced by Ariola et al. (2007). It is an extension of the call-by-value variant of $\lambda\mu$ obtained by adding a single dynamically bound continuation variable $\widehat{\mathsf{tp}}$. Ariola et al's $\lambda_{\mathcal{C}\widehat{\mathsf{tp}}}$ is a fine-grained calculus of delimited continuations in which, as an example, the semantics of Danvy and Filinski's *shift* and *reset* operators can be simulated. In the original formulation of $\lambda_{\mathcal{C}\widehat{\mathsf{tp}}}$, the control operator of the language was called $\mathcal{C}$ in spite that $\lambda_{\mathcal{C}\widehat{\mathsf{tp}}}$ is based on Parigot's structural substitution, as in $\lambda\mu$, rather than on substitution of continuations reified as ordinary functions, as it is the case in Felleisen $\lambda_{\mathcal{C}}$-calculus. Here we redefine $\lambda_{\mathcal{C}\widehat{\mathsf{tp}}}$ using the μ notation instead of a $\mathcal{C}$ notation. The so-reformulated calculus is called the call-by-value $\lambda\mu\widehat{\mathsf{tp}}$.

3.1 Syntax and reduction rules

$$
\begin{array}{rcll}
M, N & ::= & V \mid M\,M \mid \mu q.c & \text{(terms)}\\
V & ::= & x \mid \lambda x.\,M & \text{(values)}\\
c & ::= & [q]\,M & \text{(commands)}\\
q & ::= & \alpha \mid \widehat{\mathsf{tp}} & \text{(ev. context variables)}
\end{array}
$$

Figure 3. Syntax of $\lambda\mu\widehat{\mathsf{tp}}$

The syntax of $\lambda\mu\widehat{\mathsf{tp}}$ is given in Figure 3. We also define call-by-value evaluation contexts by

$$
\begin{array}{rcll}
E_v & ::= & \Box \mid E_v[\Box\,M] \mid E_v[V\,\Box] & \text{(eval. contexts)}\\
D_v & ::= & \Box \mid D_v[[q]E_v[\mu\widehat{\mathsf{tp}}.\Box]] & \text{(nested eval. context)}
\end{array}
$$

and the notations $E_v[M]$ and $D_v[c]$ stand for the terms and commands obtained by plugging M into E_v and c into D_v seen as expressions with one place-holder.

The reduction semantics of call-by-value $\lambda\mu\widehat{\mathsf{tp}}$ is given in Figure 4. The notation $c[[q]E/\alpha]$ stands for structural substitution of evaluation contexts as in $\lambda\mu$ (see Section 2). The substitutions are capture-free for term and continuation variables but $\widehat{\mathsf{tp}}$ gets captured (e.g. the substitution of x by $h\ (\mu\delta.[\widehat{\mathsf{tp}}]y)\ (\mu\delta.[\beta]z)$ in $[\alpha](f\ \mu\beta.(g\ \mu\widehat{\mathsf{tp}}.[\beta](g\ \mu\gamma.[\widehat{\mathsf{tp}}]x)))$ gives as result the expression $[\alpha](f\ \mu\beta'.(g\ \mu\widehat{\mathsf{tp}}.[\beta'](g\ \mu\gamma.[\widehat{\mathsf{tp}}](h\ (\mu\delta.[\widehat{\mathsf{tp}}]y)\ (\mu\delta.[\beta]z))))))$.

$$
\begin{array}{llcll}
\beta_v : & (\lambda x.\,M)\ V & \rightarrow & M[V/x] & \\
\mu_{app} : & (\mu\alpha.\,c)\ M & \rightarrow & \mu\beta.c[[\beta](\Box\ M)/\alpha] & \beta \text{ fresh}\\
\mu'_{app} : & V\ (\mu\alpha.\,c) & \rightarrow & \mu\beta.c[[\beta](V\ \Box)/\alpha] & \beta \text{ fresh}\\
\mu_{var} : & [q]\mu\alpha.c & \rightarrow & c[q/\alpha] & \\
\eta_{\widehat{\mathsf{tp}}} : & \mu\widehat{\mathsf{tp}}.[\widehat{\mathsf{tp}}]\ V & \rightarrow & V & \text{even if } \widehat{\mathsf{tp}} \text{ occurs in } V
\end{array}
$$

Figure 4. Reductions of call-by-value $\lambda\mu\widehat{\mathsf{tp}}$

A simple analysis of the syntax and rules shows that the unique context lemma (see Felleisen and Friedman 1986) holds: any closed command which is not of the form $[\widehat{\mathsf{tp}}]\lambda x.M$ has a unique decomposition as $D_v[[\widehat{\mathsf{tp}}]E_v[M]]$ where either M or $[\widehat{\mathsf{tp}}]E_v[M]$ is a redex. Hence the reduction system is complete for the evaluation of closed programs. A term that contains no redex at all is said to be *normal*.

3.2 Equational theory

$$
\begin{array}{llcl}
\beta_v : & (\lambda x.\,M)\ V & = & M[V/x]\\
\mu_v : & [q](E_v[\mu\alpha.\,c]) & = & c[[q]E_v/\alpha]\\
\eta_{\widehat{\mathsf{tp}}} : & \mu\widehat{\mathsf{tp}}.[\widehat{\mathsf{tp}}]\ V & = & V\\
\mu_{\widehat{\mathsf{tp}}} : & [\widehat{\mathsf{tp}}]\ (\mu\widehat{\mathsf{tp}}.c) & = & c\\
\mu^{let}_{\widehat{\mathsf{tp}}} : & \mu\widehat{\mathsf{tp}}.[q]((\lambda x.M)\ (\mu\widehat{\mathsf{tp}}.c)) & = & (\lambda x.\mu\widehat{\mathsf{tp}}.[q]M)\ (\mu\widehat{\mathsf{tp}}.c)\\
\mu^{let} : & \mu\alpha.[q]((\lambda x.N)\ M) & = & (\lambda x.\mu\alpha.[q]N)\ M^{\ 1}\\
\eta_\mu : & \mu\alpha.[\alpha]\ M & = & M^{\ 1}\\
\eta_v : & \lambda x.(V\ x) & = & V^{\ 2}\\
\beta_\Omega : & (\lambda x.E_v[x])\ M & = & E_v[M]^{\ 3}
\end{array}
$$

[1] α not free in M [2] x not free in V [3] x not free in E_v

Figure 5. CPS-complete theory of call-by-value $\lambda\mu\widehat{\mathsf{tp}}$

The equational theory of $\lambda\mu\widehat{\mathsf{tp}}$ is given in Figure 5. Note that the equation (μ_v) generalises the effect of the rules (μ_{app}), (μ'_{app}) and (μ_{var}) (up to the use of (η_μ)).

3.3 Simple typing

$$
\begin{array}{rcl}
 & X \in \mathit{TypeConstants} & \\
A, B, T, U & ::= & X \mid A_T \rightarrow B_U\\
\Gamma & ::= & \cdot \mid \Gamma, x : A\\
\Delta & ::= & \cdot \mid \Delta, \alpha : A_T
\end{array}
$$

$$\frac{}{\Gamma, x : A \vdash x : A_T; \Delta; \widehat{\mathsf{tp}} : T}\ Ax$$

$$\frac{\Gamma, x : A \vdash M : B_U; \Delta; \widehat{\mathsf{tp}} : T}{\Gamma \vdash \lambda x.M : (A_T \rightarrow B_U)_{T'}; \Delta; \widehat{\mathsf{tp}} : T'}\ \rightarrow_i$$

$$\frac{\Gamma \vdash M : (A_{T_1} \rightarrow B_{U_2})_{U_1}; \Delta; \widehat{\mathsf{tp}} : T_2 \quad \Gamma \vdash N : A_{T_1}; \Delta; \widehat{\mathsf{tp}} : U_1}{\Gamma \vdash MN : B_{U_2}; \Delta; \widehat{\mathsf{tp}} : T_2}\ \rightarrow_e$$

$$\frac{\Gamma \vdash c : \perp\!\!\!\perp; \Delta, \alpha : A_U; \widehat{\mathsf{tp}} : T}{\Gamma \vdash \mu\alpha.c : A_U; \Delta; \widehat{\mathsf{tp}} : T} \qquad \frac{\Gamma \vdash c : \perp\!\!\!\perp; \Delta; \widehat{\mathsf{tp}} : A}{\Gamma \vdash \mu\widehat{\mathsf{tp}}.c : A_T; \Delta; \widehat{\mathsf{tp}} : T}$$

$$\frac{\Gamma \vdash M : A_U; \Delta, \alpha : A_U; \widehat{\mathsf{tp}} : T}{\Gamma \vdash [\alpha]M : \perp\!\!\!\perp; \Delta, \alpha : A_U; \widehat{\mathsf{tp}} : T} \qquad \frac{\Gamma \vdash M : U_U; \Delta; \widehat{\mathsf{tp}} : T}{\Gamma \vdash [\widehat{\mathsf{tp}}]M : \perp\!\!\!\perp; \Delta; \widehat{\mathsf{tp}} : T}$$

Figure 6. Simple typing of call-by-value $\lambda\mu\widehat{\mathsf{tp}}$-calculus

The calculus $\lambda\mu\widehat{\mathsf{tp}}$ is basically an untyped calculus. Still, it is possible to constrain it with a type system. Ariola et al's adaptation of Danvy and Filinski's system of simple types (Danvy and Filinski 1989) is given in Figure 6. As in Parigot, the typing context of continuation variables is on the right of the sequent. We use the symbol $\perp\!\!\!\perp$ in the typing judgements of commands to emphasise that they have no type.

A continuation of type A_T is a continuation whose own continuation is a call to a toplevel continuation $\widehat{\mathsf{tp}}$ expected of type T, i.e. whose own continuation is expected to be called in a context where the surrounding $\mu\widehat{\mathsf{tp}}$ has type T. A judgement $\Gamma \vdash M : A_T; \Delta; \widehat{\mathsf{tp}} : U$ says that M is a term which expects a continuation of type A_T: the possible capture by M of its evaluation context will be dispatched in contexts where the dynamically closest surrounding $\mu\widehat{\mathsf{tp}}$ is of type T. In the judgement, U is the type of the actual closest surrounding toplevel $\mu\widehat{\mathsf{tp}}$. To propagate the type information of the dynamically bound $\widehat{\mathsf{tp}}$, arrows have effects: a term of type $A_T \to B_U$ is a term that expects a value of type A and returns a code of type B which:

- may capture its surrounding context and move it in a place where the toplevel has type U,
- eventually itself calls the toplevel continuation with a value of type T.

3.4 Continuation-passing-style semantics

$$
\begin{array}{lcl}
x^+ & \triangleq & x \\
(\lambda x.\, M)^+ & \triangleq & \lambda k.\lambda \nu.\lambda x.\, [\![M]\!]\; k\; \nu \\
\\
[\![V]\!]\; k\; \nu & \triangleq & k\; \nu\; V^+ \\
[\![M\; N]\!]\; k\; \nu & \triangleq & [\![M]\!]\; (\lambda \nu'.\lambda f.\, [\![N]\!]\; (f\; k)\; \nu')\; \nu \\
[\![\mu\alpha.c]\!]\; k\; \nu & \triangleq & ([\![c]\!]\; \nu)[k/\alpha] \\
[\![\mu\widehat{\mathsf{tp}}\; c]\!]\; k\; \nu & \triangleq & [\![c]\!]\; (k\; \nu) \\
[\![[\widehat{\mathsf{tp}}]\; M]\!]\; \nu & \triangleq & [\![M]\!]\; (\lambda k.\; k)\; \nu \\
[\![[\alpha]M]\!]\; \nu & \triangleq & [\![M]\!]\; \alpha\; \nu
\end{array}
$$

Figure 7. Call-by-value CPS translation of $\lambda\mu\widehat{\mathsf{tp}}$

We give in Figure 7 a continuation-passing-style (CPS) semantics in Fischer style (see Fischer 1972) for call-by-value $\lambda\mu\widehat{\mathsf{tp}}$. Ariola et al. (2007)) showed that this CPS translation can be factorised (up to currying and η-conversion) as the composition first of a state-passing-style transformation to call-by-value $\lambda\mu$ with subtraction, then of a standard call-by-value CPS translation to λ-calculus with pairs.

3.5 Equational correspondence with Kameyama and Hasegawa's axiomatisation of a calculus with *shift* and *reset*

Danvy and Filinski originally defined the operators *shift* and *reset* by their continuation-passing-style semantics. We show in this section that call-by-value $\lambda\mu\widehat{\mathsf{tp}}$ contains *shift* and *reset* in the sense that they contain operators of which the CPS semantics is the defining semantics of *shift* and *reset*.

In a second step, we show that call-by-value $\lambda\mu\widehat{\mathsf{tp}}$ contains no more than call-by-value λ-calculus extended with *shift* and *reset*. This is shown by exhibiting an equational correspondence with Kameyama and Hasegawa's theory of call-by-value λ-calculus with *shift* and *reset*, a theory known to exactly capture the CPS semantics of λ-calculus with *shift* and *reset* (Kameyama and Hasegawa 2003).

The operators *shift* and *reset* are defined as follows:

$$
\begin{array}{lcl}
\mathcal{S}\; M & \triangleq & \mu\alpha.[\widehat{\mathsf{tp}}](M\; \lambda x.\mu\widehat{\mathsf{tp}}.[\alpha]x) \\
\langle M \rangle & \triangleq & \mu\widehat{\mathsf{tp}}.[\widehat{\mathsf{tp}}]M
\end{array}
$$

The justification that these encodings define *shift* and *reset* is given by the following proposition taken from Ariola et al. (2007):

PROPOSITION 3 (Simulation of *shift* and *reset* in $\lambda\mu\widehat{\mathsf{tp}}$). *The CPS semantics of $\mathcal{S}\; M$ and $\langle M\rangle$ are:*

$$
\begin{array}{lcl}
[\![\mathcal{S}\; (\lambda q.\; M)]\!]\; k\; \nu & = & [\![M]\!]\; [\lambda k'.\lambda \nu.\; k\; (k'\; \nu)/q]\; (\lambda k.k)\; \nu \\
[\![\langle M\rangle]\!]\; k\; \nu & = & [\![M]\!]\; (\lambda k.k)\; (k\; \nu)
\end{array}
$$

which coincide with the defining CPS semantics of shift and reset in Danvy and Filinski (1989).

Let now $(\lambda_{\mathcal{S}}, =_{KH})$ be λ-calculus equipped with *shift* and *reset* and with the axiomatics of Kameyama and Hasegawa (2003). Let $(\lambda\mu\widehat{\mathsf{tp}}, =)$ be call-by-value $\lambda\mu\widehat{\mathsf{tp}}$ equipped with the axioms given in Figure 5. Let $\lambda\mu\widehat{\mathsf{tp}}^0$ be the subset of expressions of $\lambda\mu\widehat{\mathsf{tp}}$ that do not contain free continuation variables. We define $(\lambda\mu\widehat{\mathsf{tp}}^0, =)$ as the restriction of $(\lambda\mu\widehat{\mathsf{tp}}, =)$ to the expressions of $\lambda\mu\widehat{\mathsf{tp}}^0$.

The interpretation of $\lambda\mu\widehat{\mathsf{tp}}$ in Kameyama and Hasegawa's calculus works as follows: each continuation variable α is injectively mapped to a fresh ordinary variable k_α, $\mu\widehat{\mathsf{tp}}.[\widehat{\mathsf{tp}}]t$ is interpreted as $\langle M\rangle$, $\mu\widehat{\mathsf{tp}}.[\alpha]t$ as $\langle k_\alpha\; M\rangle$, $\mu\alpha.[\beta]M$ as $\mathcal{S}(\lambda k_\alpha.k_\beta\;\; M)$ and $\mu\alpha.[\widehat{\mathsf{tp}}]M$ as $\mathcal{S}(\lambda k_\alpha.M)$. The next theorem expresses the equational correspondence (in the sense of Sabry and Felleisen 1993) between call-by-value $\lambda\mu\widehat{\mathsf{tp}}^0$ and Kameyama and Hasegawa's calculus:

THEOREM 4 (Ariola et al. 2007). *The theories $(\lambda_{\mathcal{S}}, =_{KH})$ and $(\lambda\mu\widehat{\mathsf{tp}}^0, =)$ are isomorphic.*

COROLLARY 5 (Ariola et al. 2007). *The theory $(\lambda\mu\widehat{\mathsf{tp}}, =)$ is complete with respect to β and η through the CPS semantics of call-by-value $\lambda\mu\widehat{\mathsf{tp}}$.*

The addition of a continuation delimiter was used in Sitaram and Felleisen (1990) to recover some completeness property that was lost in the move from λ-calculus to λ_C-calculus. Our analysis of Saurin's separability result for call-by-name $\lambda\mu$ in Section 4.3 shows that Böhm's theorem, which amounts to observational completeness for normal terms, is also recovered by the addition of a continuation delimiter. This suggests the following conjecture:

CONJECTURE 6. *The theory $(\lambda\mu\widehat{\mathsf{tp}}, =)$ satisfies Böhm's theorem, i.e., for any equationally distinct closed normal forms M and N, there is a context $D_v[[q]E_v]$ such that $\mu\widehat{\mathsf{tp}}.D_v[[q](E_v[M])] = x$ and $\mu\widehat{\mathsf{tp}}.D_v[[q](E_v[N])] = y$.*

3.6 Operational semantics

The operational semantics in "natural" style is characterised by a deterministic application of the reduction rules at the head of a computation (so-called weak-head reduction). It is common to formulate the operational semantics on terms but we rather do it on commands what allows for a more uniform characterisation of normal forms. Typically, when formulated on terms, a term like $\mu\alpha.[\alpha]V$ can be reduced further to V only if α does not occur in V but it cannot be reduced further if α does occur. To the contrary, if the same term is reduced as part of a command, as in $[\beta]\mu\alpha.[\alpha]V$, then the resulting command uniformly reduces to $[\beta](V[\beta/\alpha])$ independently of whether α occurs or not in V. The operational semantics, that we do not only define on closed terms as it is common but also on terms with free variables, is given by the equations:

$$
\begin{array}{llcl}
\beta : & D_v[[q]E_v[(\lambda x.M)\; V]] & \mapsto & D_v[[q]E_v[M[V/x]]] \\
\mu_v : & D_v[[q]E_v[\mu\beta.c]] & \mapsto & D_v[c[[q]E_v/\beta]] \\
\eta_{\widehat{\mathsf{tp}}} : & D_v[[q]E_v[\mu\widehat{\mathsf{tp}}.[\widehat{\mathsf{tp}}]V]] & \mapsto & D_v[[q]E_v[V]]
\end{array}
$$

Obviously $\mapsto$ is included in $\twoheadrightarrow$ of which it constitutes on commands a convenient level of abstraction. We say that c is a *weak-head normal* command if for no c', $c \mapsto c'$. Weak-head normal

$$
\begin{array}{lcll}
K & ::= & q[e] \mid M[e]\cdot K \mid \tilde{\mu}\square.\langle W \parallel \square\cdot K\rangle & \text{(evaluation contexts)}\\
[S] & ::= & [\,] \mid [\widehat{\mathsf{tp}} = K; S] & \text{(dynamic environment)}\\
W & ::= & V[e] & \text{(closure of value)}\\
[e] & ::= & [\,] \mid [x = W; e] \mid [\alpha = K; e] & \text{(environments)}\\
s & ::= & \langle W \mid\mid K\rangle_{\mathit{cont}}\,[S] \mid \langle M\,[e] \mid\mid K\rangle_{\mathit{eval}}\,[S] \mid \langle V\,[e] \mid\mid W\cdot K\rangle_{\mathit{logic}}\,[S] & \text{(states)}
\end{array}
$$

Figure 8. Specific components of the abstract machine for call-by-value $\lambda\mu\widehat{\mathsf{tp}}$-calculus

Control owned by the evaluation context

$$
\begin{array}{llllcllll}
\langle W & \mid\mid N[e]\cdot K & \rangle_{\mathit{cont}} & [S] & \rightarrow & \langle N\ \ [e] & \mid\mid \tilde{\mu}\square.\langle W \parallel \square\cdot K\rangle & \rangle_{\mathit{eval}}\ [S] & \\
\langle W & \mid\mid \tilde{\mu}\square.\langle V\,[e] \parallel \square\cdot K\rangle & \rangle_{\mathit{cont}} & [S] & \rightarrow & \langle V\ \ [e] & \mid\mid W\cdot K & \rangle_{\mathit{logic}}\ [S] & \\
\langle W & \mid\mid \widehat{\mathsf{tp}}[e] & \rangle_{\mathit{cont}} & [\widehat{\mathsf{tp}} = K; S] & \rightarrow & \langle W & \mid\mid K & \rangle_{\mathit{cont}}\ [S] & \\
\langle W & \mid\mid \alpha[e] & \rangle_{\mathit{cont}} & [S] & \rightarrow & \langle W & \mid\mid K & \rangle_{\mathit{cont}}\ [S] & \text{if } e(\alpha) = K\\
\langle W & \mid\mid \widehat{\mathsf{tp}}[e] & \rangle_{\mathit{cont}} & [\,] & \rightarrow & \multicolumn{3}{l}{\text{stop on } [\widehat{\mathsf{tp}}]W^{\dagger}} & \\
\langle W & \mid\mid \alpha[e] & \rangle_{\mathit{cont}} & [S] & \rightarrow & \multicolumn{3}{l}{\text{stop on } S^{\dagger}[[\alpha]W^{\dagger}]} & \text{if } \alpha \text{ not bound in } e
\end{array}
$$

Control owned by the term

$$
\begin{array}{lllllcllll}
\langle V & [e] & \mid\mid K & \rangle_{\mathit{eval}} & [S] & \rightarrow & \langle V[e] & & \mid\mid K & \rangle_{\mathit{cont}}\ [S]\\
\langle M\,N & [e] & \mid\mid K & \rangle_{\mathit{eval}} & [S] & \rightarrow & \langle M & [e] & \mid\mid N[e]\cdot K & \rangle_{\mathit{eval}}\ [S]\\
\langle \mu\alpha.[q]M & [e] & \mid\mid K & \rangle_{\mathit{eval}} & [S] & \rightarrow & \langle M & [\alpha = K; e] & \mid\mid q[\alpha = K; e] & \rangle_{\mathit{eval}}\ [S]\\
\langle \mu\widehat{\mathsf{tp}}.[q]M & [e] & \mid\mid K & \rangle_{\mathit{eval}} & [S] & \rightarrow & \langle M & [e] & \mid\mid q[e] & \rangle_{\mathit{eval}}\ [\widehat{\mathsf{tp}} = K; S]
\end{array}
$$

Control owned by the value

$$
\begin{array}{lllllcllll}
\langle \lambda x.M & [e] & \mid\mid W\cdot K & \rangle_{\mathit{logic}} & [S] & \rightarrow & \langle M & [x = W; e] & \mid\mid K & \rangle_{\mathit{eval}}\ [S]\\
\langle x & [e] & \mid\mid W\cdot K & \rangle_{\mathit{logic}} & [S] & \rightarrow & \langle V & [e'] & \mid\mid W\cdot K & \rangle_{\mathit{logic}}\ [S] \quad \text{if } e(x) = V[e']\\
\langle x & [e] & \mid\mid W\cdot K & \rangle_{\mathit{logic}} & [S] & \rightarrow & \multicolumn{3}{l}{\text{stop on } S^{\dagger}[K^{\dagger}[x\ W^{\dagger}]]} & \text{if } x \text{ not bound in } e
\end{array}
$$

To evaluate M, the machine starts with the following initial state:

$$\langle M\ [\,] \mid\mid \widehat{\mathsf{tp}}[\,]\rangle_{\mathit{eval}}\ [\,]$$

Figure 9. Abstract machine for call-by-value $\lambda\mu\widehat{\mathsf{tp}}$

commands are either of the form $[\widehat{\mathsf{tp}}]V$, or of the form $D_v[[\alpha]V]$, or of the form $D_v[[q]E_v[x\ V]]$.

Operational semantics can be also described by using an abstract machine. Evaluation in an abstract machine is closely related to cut-elimination in Gentzen's sequent calculus (see e.g. Herbelin 1995, 1997; Danos et al. 1996) while, contrastingly, operational semantics in "natural" style is related to Gentzen's natural deduction. Sequent calculus proofs can be represented in $\overline{\lambda}\mu\tilde{\mu}$-calculus (Curien and Herbelin 2000; Herbelin 2005) extended with a calculus of explicit substitutions (see e.g. Herbelin 2001) to represent closures and environments, in the spirit of Hardin et al. (1996). The language of the abstract machine for call-by-value $\lambda\mu\widehat{\mathsf{tp}}$ is shown in Figure 8 and the reduction steps are given in Figure 9. The syntax for evaluation contexts and states is reminiscent of $\overline{\lambda}\mu\tilde{\mu}$-calculus. Stacks are identified with evaluation contexts. The construction $q[e]$ refers to the continuation bound to q in the environment e. The construction $M[e]\cdot K$ denotes the continuation which first applies $M[e]$ before continuing with continuation K. The construction $\tilde{\mu}\square.\langle V\ [e] \parallel \square\cdot K\rangle$ denotes the continuation that binds to $\square$ the current result, say W, so that computation continues with code V in environment e and continuation $W\cdot K$. The construction $\langle M \parallel K\rangle$ denotes the interaction of a term M in context K. This construction comes in three flavours. In $\langle M\ [e] \mid\mid K\rangle_{\mathit{eval}}\ [S]$, the term M in environment e has the control on what is going next. At some point of the evaluation process, the term gets evaluated and the control is transfered to the evaluation context. This corresponds to a state $\langle W \mid\mid K\rangle_{\mathit{cont}}\ [S]$. At some point, both the term and the evaluation context are in "evaluated" form and a "logical" interaction happens. This corresponds to states of the form $\langle V\ [e] \mid\mid W\cdot K\rangle_{\mathit{logic}}\ [S]$. Specific evaluation rules correspond to each of these different states. We write $e(\alpha)$ for the binding of α in e and similarly for $e(x)$. The dynamically bound variables are bound in an environment S that remains global (it is not stored in closures). Note that when the dynamic continuation variable $\widehat{\mathsf{tp}}$ is referred to, not only the continuation bound to $\widehat{\mathsf{tp}}$ is restored but the binding is *removed* so that the next call to $\widehat{\mathsf{tp}}$ will refer to the next binding of the global environment.

Observe that the abstract machine is designed to return the weak-head normal form not only of closed programs but also of terms with free variables (see the "stop" transitions). Final result reconstruction in terminal states turns explicit substitutions into effective substitutions. Result reconstruction turns closures of values into ordinary values. It also uses the operation $S^{\dagger}$ that builds contexts for command of the form D_v and the operation $K^{\dagger}$ that builds contexts of the form $[q]E_v$. These operations are defined by the following clauses:

$$
\begin{array}{lll}
[\alpha[e]]^\dagger & \triangleq & K^\dagger \quad \text{if } e(\alpha) = K \\
[\alpha[e]]^\dagger & \triangleq & [\alpha](\Box) \quad \text{otherwise} \\
[\widehat{\mathtt{tp}}[e]]^\dagger & \triangleq & [\widehat{\mathtt{tp}}](\Box) \\
(M[e]\cdot K)^\dagger & \triangleq & K^\dagger[\Box\, M[e]^\dagger] \\
(\tilde{\mu}\Box.\langle W \parallel \Box\cdot K\rangle)^\dagger & \triangleq & K^\dagger[W^\dagger\ \Box] \\
\\
[\,]^\dagger & \triangleq & \Box \\
[\widehat{\mathtt{tp}} = K; S]^\dagger & \triangleq & S^\dagger[K^\dagger[\mu\widehat{\mathtt{tp}}.\Box]] \\
M[x = W; e]^\dagger & \triangleq & M[W^\dagger/x][e]^\dagger \\
M[\alpha = K; e]^\dagger & \triangleq & M[K^\dagger/\alpha][e]^\dagger
\end{array}
$$

PROPOSITION 7. *If c is a weak-head normal form, then, $[\widehat{\mathtt{tp}}]M \mapsto\!\!\!\rightarrow c$ iff the evaluation starting from $\langle M\ [\,]\ \|\ \widehat{\mathtt{tp}}[\,]\rangle_{eval}\ [\,]$ stops with result c.*

To make a comparison with the operational semantics of *shift* and *reset* given in Biernacka et al. (2003), we show how (the encoding of) *shift* and *reset* in $\lambda\mu\widehat{\mathtt{tp}}$ operationally reduce on closed terms. Using the abbreviation $C_v ::= D_v[[\widehat{\mathtt{tp}}]E_v]$, we get:

$$
\begin{array}{lll}
C_v[\langle E_v[\mathcal{S}(\lambda k.M)]\rangle] & \mapsto & C_v[\langle M[\lambda x.\langle E_v[x]\rangle/k]\rangle] \\
C_v[\langle V\rangle] & \mapsto & C_v[V]
\end{array}
$$

which coincide with the rules $(\mathcal{S}_\lambda)$ and (val) in Biernacka et al. (2003, Section 4.4) through the identification of $C_2\#$ with D_v and of C_1 with, $[\widehat{\mathtt{tp}}]E_v$ if in position of context, or $\lambda x.\langle E_v[x]\rangle$ if in position of term. As for the rules (β_λ) and (β_{ctx}) in Biernacka et al. (2003) they both are instances of (β).

3.7 Expressiveness

Call-by-value $\lambda\mu\widehat{\mathtt{tp}}$ is fine-grained enough to directly simulate the operational semantics of most standard control operators. Let $\mathcal{C}$ and $\mathcal{A}$ be Felleisen's $\mathcal{C}$ and *abort* operators[4]. Let `call/cc` be the implementation of `call-cc` in Scheme. Let M `handle` *patterns* and `raise` M be the constructions of the exception mechanism in SML (i.e. of `try` M `with` *patterns* and `raise` M in O'Caml). Let `Val` be a special exception with one argument. In addition to the definition of $\mathcal{S}\ M$ and $\langle M\rangle$ above, we have the following encodings:

$$
\begin{array}{lll}
\mathcal{A}\,M & \triangleq & \mu\_.[\widehat{\mathtt{tp}}]M \\
\mathcal{C}\,(\lambda k.M) & \triangleq & \mu\alpha_k.[\widehat{\mathtt{tp}}](M\ \lambda x.\mu\_.[\alpha_k]x) \\
\mathtt{call/cc}\,(\lambda k.M) & \triangleq & \mu\alpha_k.[\alpha_k](M\ \lambda x.\mu\_.[\alpha_k]x) \\
\mathtt{raise}\ M & \triangleq & \mu\_.[\widehat{\mathtt{tp}}]M \\
M\ \mathtt{handle}\ \textit{patterns} & \triangleq & \mathtt{case}\ \mu\widehat{\mathtt{tp}}.[\widehat{\mathtt{tp}}](\mathtt{Val}\ M)\ \mathtt{of} \\
 & & \mid\ \mathtt{Val}\ x\ \Rightarrow\ x \\
 & & \mid\ \textit{patterns} \\
 & & \mid\ x\ \Rightarrow\ \mu\_.[\widehat{\mathtt{tp}}]x
\end{array}
$$

Let us show for instance how the operational semantics of Scheme's `call/cc` is faithfully simulated[5]. Thanks to structural substitution, we have:

$$E_v[\mathtt{call/cc}\ \lambda k.M] \mapsto E_v[M[\lambda x.\mathcal{A}\ E_v[x]/k]]$$

while any other encoding from $\mathcal{C}$ (e.g. `call/cc` $(\lambda k.M) \triangleq \mathcal{C}(\lambda k.k\ M)$) or from $\mathcal{S}$ (e.g. `call/cc` $(\lambda k.M) \triangleq \mathcal{S}(\lambda k.k\ M')$ with $M' \triangleq M[\lambda x.\mathcal{A}\,(k\,x)/k]$) would give the following wrong semantics:

$$E_v[\mathtt{call/cc}\ \lambda k.M] \mapsto (\lambda x.\mathcal{A}\ E_v[x])\ M[\lambda x.\mathcal{A}\ E_v[x]/k]\ .$$

Structural substitution also brings new behaviours. Here are a few examples:

$$
\begin{array}{ll}
(i): & \mu\alpha.[\alpha](\ldots\ \mu\_.[\alpha]M\ \ldots\ \mu\_.[\alpha]M'\ \ldots) \\
(ii): & \mu\alpha.[\widehat{\mathtt{tp}}](\ldots\ \mu\widehat{\mathtt{tp}}.[\alpha]M\ \ldots\ \mu\widehat{\mathtt{tp}}.[\alpha]M'\ \ldots) \\
(iii): & \mu\alpha.[\widehat{\mathtt{tp}}](\ldots\ \mu\widehat{\mathtt{tp}}.[\alpha]M\ \ldots\ \mu\_.[\alpha]M'\ \ldots) \\
(iv): & \mu\alpha.[\widehat{\mathtt{tp}}](\ldots\ \mu\beta.[\alpha](\ldots\ [\beta]M\ \ldots\ [\alpha]M'\ \ldots)\ \ldots)
\end{array}
$$

Example (i) is an "optimised" variant of `call/cc` that does not need to wait that the argument of the continuation is evaluated before to reinstall the continuation (compare $(\lambda x.\mu\_.[\alpha]x)\ M$ with $\mu\_.[\alpha]M$). Example (ii) is a similarly "optimised" variant of *shift*. Example (iii) is an hybrid operator which is compositional on the left call to α (like *shift*) and abortive on the right call to α (like $\mathcal{C}$). Finally, example (iv) shows how a call to continuation α in the body of a call to another control operator can be contracted (compare $\mu\beta.[\widehat{\mathtt{tp}}]((\lambda x.\mu\_[\alpha]x)\ M)$, as in, e.g., the interpretation of $\mathcal{S}(\lambda k_\beta.k_\alpha\ M)$, with $\mu\beta.[\alpha]M$). More generally, see Ariola and Herbelin (2007) for an analysis of the advantages of structural substitution of evaluation contexts over substitution of continuations as regular functions.

Thanks to Filinski's result on the ability of *shift* and *reset* to encode monads (Filinski 1994), one can also simulate, as an example, reading and writing to a memory cell:

$$
\begin{array}{lll}
\mathtt{read} & \triangleq & \lambda().\mu\alpha.[\widehat{\mathtt{tp}}]\lambda s.((\mu\widehat{\mathtt{tp}}.[\alpha]s)\ s) \\
\mathtt{write} & \triangleq & \lambda s.\mu\alpha.[\widehat{\mathtt{tp}}]\lambda\_.((\mu\widehat{\mathtt{tp}}.[\alpha]())\ s)
\end{array}
$$

so that the code $(\mu\widehat{\mathtt{tp}}.[\widehat{\mathtt{tp}}]M)\ v_0$ which may refer to `read` and `write` behaves as a program M reading and writing to a global memory cell initialised to value v_0.

4. Call-by-Name $\lambda\mu\widehat{\mathtt{tp}}$-Calculus

In this section we introduce a call-by-name variant of $\lambda\mu\widehat{\mathtt{tp}}$. We formalise a reduction semantics, an equational theory, a system of simple types, a continuation-passing-style semantics and an operational semantics. Call-by-name $\lambda\mu\widehat{\mathtt{tp}}$ is an extension of $\lambda\mu$ that we show to be isomorphic to $\Lambda\mu$. From the programming point of view, call-by-name $\lambda\mu\widehat{\mathtt{tp}}$, is a bizarre calculus. In an attempt to clarify how it behaves, we end the section by an example.

4.1 Syntax and reduction rules

The syntax of $\lambda\mu\widehat{\mathtt{tp}}$ was given in Figure 3. The reduction rules of the call-by-name variant of $\lambda\mu\widehat{\mathtt{tp}}$ are in Figure 10. They extend the reduction rules of $\lambda\mu$ in Figure 1 by one rule, called $(\eta_{\widehat{\mathtt{tp}}})$, that is similar to the equation (η_μ) but without any constraints on whether $\widehat{\mathtt{tp}}$ occurs or not in M.

$$
\begin{array}{llll}
\beta: & (\lambda x.\, M)\ N & \to & M[N/x] \\
\mu_{app}: & (\mu\alpha.\, c)\ M & \to & \mu\beta.c[[\beta](\Box\ M)/\alpha] \qquad \beta \text{ fresh} \\
\mu^n_{var}: & [\beta]\mu\alpha.c & \to & c[\beta/\alpha] \\
\eta^n_{\widehat{\mathtt{tp}}}: & \mu\widehat{\mathtt{tp}}.[\widehat{\mathtt{tp}}]\ M & \to & M \qquad \text{even if } \widehat{\mathtt{tp}} \text{ occurs in } M
\end{array}
$$

Figure 10. Reductions of call-by-name $\lambda\mu\widehat{\mathtt{tp}}$

We say that a term or a command is *normal* if it contains no redex. Call-by-name evaluation contexts are defined as follows:

$$
\begin{array}{llll}
E_n & ::= & \Box \mid E_n[\Box\ M] & \text{(linear eval. contexts)} \\
D_n & ::= & \Box \mid D_n[[\alpha]E_n[\mu\widehat{\mathtt{tp}}.\Box]] & \text{(nested linear eval. context)}
\end{array}
$$

[4] As usual, $\mathcal{A}$ can be used itself to simulate `break` or `return` in imperative language, assuming that a $\langle\,\rangle$ marker has been inserted around the related block.

[5] Note that the SML variant of `call/cc` is not directly simulatable as it reifies the whole undelimited continuation including the exception handlers, which would mean that $\lambda\mu\widehat{\mathtt{tp}}$ semantics would have to support the capture of the $\mu\widehat{\mathtt{tp}}$ markers, what it does not. Hence, only the Scheme's variant of `call/cc`, which does not interfere with exception handling, is simulatable.

Call-by-name evaluation contexts are called *linear* because they do not erase nor duplicate the term they expect in their hole.

An analysis of the syntax and rules shows that a form of unique context lemma holds: any command with α as only free variable is either of the form $D_n[[\alpha]\lambda x.M]$, or of the form $[\widehat{\mathsf{tp}}]M$, or, of the form $D_n[[\alpha](E_n[M])]$ with either M or $[\alpha](E_n[M])$ a redex. This is in fact a curious result: $\widehat{\mathsf{tp}}$ blocks the reduction and there is not much hope to compute something interesting without at least one free continuation variable at hand.

4.2 Equational theory

The equational theory of call-by-name $\lambda\mu\widehat{\mathsf{tp}}$ extends the equational theory of $\lambda\mu$ with $(\mu_{\widehat{\mathsf{tp}}})$ (an analog of the rule (μ_{var}) for the special continuation $\widehat{\mathsf{tp}}$) and with $(\eta_{\widehat{\mathsf{tp}}})$ seen as equation. It is given in Figure 11.

$$
\begin{array}{lllll}
\beta: & (\lambda x.M)\,N & = & M[N/x] & \\
\mu_n: & [\beta](E_n[\mu\alpha.c]) & = & c[[\beta]E_n/\alpha] & \\
\eta_\mu: & \mu\alpha.[\alpha]M & = & M & \text{if } \alpha \text{ not free in } M \\
\eta: & \lambda x.(M\,x) & = & M & \text{if } x \text{ not free in } M \\
\mu_{\widehat{\mathsf{tp}}}: & [\widehat{\mathsf{tp}}]\mu\widehat{\mathsf{tp}}.c & = & c & \\
\eta_{\widehat{\mathsf{tp}}}: & \mu\widehat{\mathsf{tp}}.[\widehat{\mathsf{tp}}]M & = & M &
\end{array}
$$

Figure 11. Equational theory of call-by-name $\lambda\mu\widehat{\mathsf{tp}}$-calculus

4.3 Equational correspondence with $\Lambda\mu$

The calculus $\Lambda\mu$ is derived from $\lambda\mu$ by relaxing the syntax and keeping the same theory. We now show how $\Lambda\mu$ can be contrastingly restated as a strict extension of $\lambda\mu$. This extension is precisely our call-by-name variant of $\lambda\mu\widehat{\mathsf{tp}}$.

Embedding of $\Lambda\mu$ into call-by-name $\lambda\mu\widehat{\mathsf{tp}}$ A naive way to interpret $\Lambda\mu$ in $\lambda\mu$, actually in its extension $\lambda\mu\widehat{\mathsf{tp}}$, is to interpret any $\Lambda\mu$-term $\mu\alpha.M$ as the $\lambda\mu$-term $\mu\alpha.[\widehat{\mathsf{tp}}]M$ and to interpret any $\Lambda\mu$-term $[\alpha]M$ as the $\lambda\mu$-term $\mu\widehat{\mathsf{tp}}.[\alpha]M$. Formally, this corresponds to the following embedding Π of $\Lambda\mu$ into $\lambda\mu\widehat{\mathsf{tp}}$:

$$
\begin{array}{lll}
\Pi(x) & \triangleq & x \\
\Pi(\lambda x.M) & \triangleq & \lambda x.\Pi(M) \\
\Pi(M\,N) & \triangleq & \Pi(M)\,\Pi(N) \\
\Pi(\mu\alpha.M) & \triangleq & \mu\alpha.[\widehat{\mathsf{tp}}]\Pi(M) \\
\Pi([\alpha]M) & \triangleq & \mu\widehat{\mathsf{tp}}.[\alpha]\Pi(M)
\end{array}
$$

This translation is not defined on continuation variables, since they are not part of the formal syntax. Nevertheless we can derive the following property:

LEMMA 8.

$$\Pi(M[\beta/\alpha]) = \Pi(M)[\beta/\alpha]$$

We then check that all rules of $\Lambda\mu$ can be simulated in $\lambda\mu$, all but the (μ_{var}) rule. Indeed,

$$
\begin{array}{lll}
\Pi([\beta]\mu\alpha.M) & \equiv & \mu\widehat{\mathsf{tp}}.[\beta]\mu\alpha.[\widehat{\mathsf{tp}}]\Pi(M) \\
& \rightarrow_{\mu_{var}} & \mu\widehat{\mathsf{tp}}.[\widehat{\mathsf{tp}}]\Pi(M)[\beta/\alpha] \\
& \equiv & \mu\widehat{\mathsf{tp}}.[\widehat{\mathsf{tp}}]\Pi(M[\beta/\alpha])
\end{array}
$$

but $\mu\widehat{\mathsf{tp}}.[\widehat{\mathsf{tp}}].\Pi(M[\beta/\alpha])$ has no reason to be equal to $\Pi(M[\beta/\alpha])$ in $\lambda\mu$. This is actually expected since $\Lambda\mu$ is observationally complete for normal terms but $\lambda\mu$ is not. However, in the extended calculus $\lambda\mu\widehat{\mathsf{tp}}$, this equality holds. Indeed, we now have:

PROPOSITION 9. *If $M = N$ in $\Lambda\mu$ then $\Pi(M) = \Pi(N)$ in $\lambda\mu\widehat{\mathsf{tp}}$.*

Embedding of call-by-name $\lambda\mu\widehat{\mathsf{tp}}$ into $\Lambda\mu$ We now want to show that our call-by-name $\lambda\mu\widehat{\mathsf{tp}}$, i.e. $\lambda\mu$ extended with rules $(\mu_{\widehat{\mathsf{tp}}})$ and $(\eta_{\widehat{\mathsf{tp}}})$, is indeed equivalent to $\Lambda\mu$. Let us define the following converse translation:

$$
\begin{array}{lll}
\Sigma(x) & \triangleq & x \\
\Sigma(\lambda x.M) & \triangleq & \lambda x.\Sigma(M) \\
\Sigma(M\,N) & \triangleq & \Sigma(M)\,\Sigma(N) \\
\Sigma(\mu\alpha.[\beta]M) & \triangleq & \mu\alpha.([\beta]\Sigma(M)) \\
\Sigma(\mu\alpha.[\widehat{\mathsf{tp}}]M) & \triangleq & \mu\alpha.(\Sigma(M)) \\
\Sigma(\mu\widehat{\mathsf{tp}}.[\alpha]M) & \triangleq & [\alpha]\Sigma(M) \\
\Sigma(\mu\widehat{\mathsf{tp}}.[\widehat{\mathsf{tp}}]M) & \triangleq & \Sigma(M)
\end{array}
$$

PROPOSITION 10. *If $M = N$ in call-by-name $\lambda\mu\widehat{\mathsf{tp}}$, then $\Sigma(M) = \Sigma(N)$ in $\Lambda\mu$.*

Since moreover, $\Sigma(\Pi(M)) \equiv M$ and $\Pi(\Sigma(M)) = M$ (using $(\mu_{\widehat{\mathsf{tp}}})$), we get:

THEOREM 11 (Equational correspondence). *$\Lambda\mu$ equipped with the equations of Figure 2 and call-by-name $\lambda\mu\widehat{\mathsf{tp}}$ equipped with the equations of Figure 11 equationally correspond.*

REMARK: Call-by-name $\lambda\mu\widehat{\mathsf{tp}}$ and $\Lambda\mu$ form more than an equational correspondence: their reduction systems are also bisimilar: $M \twoheadrightarrow N$ iff $\Pi(M) \twoheadrightarrow \Pi(N)$ and $\Sigma(M) \twoheadrightarrow \Sigma(N)$ iff $M \twoheadrightarrow N$. In particular, normal forms match.

Observational completeness of normal forms in $\lambda\mu\widehat{\mathsf{tp}}$ As a consequence of the isomorphism, we have:

COROLLARY 12. *Call-by-name $\lambda\mu\widehat{\mathsf{tp}}$ is observationally complete for closed normal forms: Calling a normal form* canonical *if it contains no subterms of the form $[\alpha]\lambda x.M$, we have that for any closed canonical normal forms M and N not equal in call-by-name $\lambda\mu\widehat{\mathsf{tp}}$, there exists an evaluation context D_n, such that, in $\lambda\mu\widehat{\mathsf{tp}}$, $\mu\widehat{\mathsf{tp}}.D_n[[\widehat{\mathsf{tp}}]M] = x$ and $\mu\widehat{\mathsf{tp}}.D_n[[\widehat{\mathsf{tp}}]N] = y$ for x and y arbitrary fresh variables.*

Interestingly, this shows that if Böhm's theorem in $\Lambda\mu$ (Theorem 2) was apparently obtained by allowing more contexts (namely contexts of the form $\Box\ [\alpha]M$) which were not allowed in Parigot's syntax, it is alternatively obtained by adding not only more contexts but by adding new rules that were hidden by the fact that $\lambda\mu$ and $\Lambda\mu$ apparently share the same rules.

One may wonder whether the equational theory of call-by-name $\lambda\mu\widehat{\mathsf{tp}}$ is complete with respect to its CPS semantics. This is answered positively in Section 4.5.

4.4 Simple typing

We propose a system of simple types for call-by-name $\lambda\mu\widehat{\mathsf{tp}}$. Like for typing $\lambda\mu$, we have two kinds of sequents, one for each category of expressions:

$$
\begin{array}{ll}
\Gamma \vdash_\Sigma M : A; \Delta & \text{(for terms)} \\
\Gamma \vdash_\Sigma c : \bot\!\!\!\bot; \Delta & \text{(for commands)}
\end{array}
$$

Like for $\lambda\mu$, we have a context of *hypotheses* Γ that assigns types to term variables and a context of *conclusions* Δ that assigns types to continuation variables. But we have also to take care of the $\mu\widehat{\mathsf{tp}}$ dynamic binder.

Like for Ariola et al's adaptation to call-by-value $\lambda\mu\widehat{\mathsf{tp}}$ of Danvy and Filinski's typing system in Section 3.3, we have an extra data to type the dynamic effects. Each use of $\mu\widehat{\mathsf{tp}}$ pushes the current continuation on a stack of dynamically bound continuations. Each call to $\widehat{\mathsf{tp}}$ pops the top continuation from this stack.

To the contrary of Ariola et al's typing system, the extra information needed to type the dynamic binding is not a single formula

$$
\begin{array}{rcl}
X & \in & \textit{TypeConstants} \\
A, B & ::= & X \mid A_\Sigma \to B \\
\Gamma & ::= & \emptyset \mid \Gamma, x : A_\Sigma \\
\Delta & ::= & \emptyset \mid \Delta, \alpha : A \\
\Sigma, \Xi & ::= & \perp \mid A \cdot \Sigma
\end{array}
$$

$$
\frac{}{\Gamma, x : A_\Sigma \vdash_\Sigma x : A; \Delta}\ Ax
$$

$$
\frac{\Gamma, x : A_\Sigma \vdash_\Xi M : B; \Delta}{\Gamma \vdash_\Xi \lambda x.M : (A_\Sigma \to B); \Delta}\ \to_i
$$

$$
\frac{\Gamma \vdash_\Xi M : (A_\Sigma \to B); \Delta \quad \Gamma \vdash_\Sigma N : A; \Delta}{\Gamma \vdash_\Xi M\, N : B; \Delta}\ \to_e
$$

$$
\frac{\Gamma \vdash_\Sigma c : \perp\!\!\!\perp; \Delta, \alpha : A}{\Gamma \vdash_\Sigma \mu\alpha.c : A; \Delta}
\qquad
\frac{\Gamma \vdash_{A\cdot\Sigma} c : \perp\!\!\!\perp; \Delta}{\Gamma \vdash_\Sigma \mu\widehat{\mathsf{tp}}.c : A; \Delta}
$$

$$
\frac{\Gamma \vdash_\Sigma M : A; \Delta, \alpha : A}{\Gamma \vdash_\Sigma [\alpha]M : \perp\!\!\!\perp; \Delta, \alpha : A}
\qquad
\frac{\Gamma \vdash_\Sigma M : A; \Delta}{\Gamma \vdash_{A\cdot\Sigma} [\widehat{\mathsf{tp}}]M : \perp\!\!\!\perp; \Delta}
$$

Figure 12. Simple typing of call-by-name $\lambda\mu\widehat{\mathsf{tp}}$-calculus

but the ordered list Σ of the types of the continuations present in the stack.

Like for Ariola et al's typing system, functions can encapsulate occurrences of $\widehat{\mathsf{tp}}$ that may be called in a different typing context than the one that was active at the time $\mu\widehat{\mathsf{tp}}$ was typed. For type consistency, arrows have to remember the types of the dynamic continuation stack that the calls to $\widehat{\mathsf{tp}}$ expect to see. We write $A_\Sigma \to B$ for an arrow annotated with the list Σ of effect types.

To the contrary of Ariola et al's typing system, calls to $\widehat{\mathsf{tp}}$ are associated to terms and hence effects are assigned to the types of Γ rather than to the types of Δ. The typing system is given in Figure 12.

A very similar system of simple types has been given by Saurin (2007) on top of $\Lambda\mu$. In $\Lambda\mu$, judgements $\Gamma \vdash_\Sigma c : \perp\!\!\!\perp; \Delta$ are absent since there are no commands in the calculus. Judgements $\Gamma \vdash_\Sigma M : A; \Delta$ are written $\Gamma, \Delta \vdash M : A \Rightarrow \Sigma$. Moreover, $A_1 \cdot \ldots \cdot A_n$ is written $A_1 \Rightarrow \ldots \Rightarrow A_n \Rightarrow o$ and $A_\Sigma \to B$ is written $(A \Rightarrow \Sigma) \to B$. Intuitively, $A \Rightarrow B$ denotes a term that returns an object of type B when applied to a linear evaluation context of type A (a *stream* in Saurin's terminology). Logically, the type $A \Rightarrow B$ is equivalent to $\neg A \to B$, where the use of a negation emphasises that $\neg A$ is the type of an evaluation context expecting an argument of type A. Hence, $A \Rightarrow B$ is logically equivalent to a disjunction. Note that because $\Rightarrow$ is a connective in Saurin's typing system, a conversion rule from $((A \Rightarrow \Sigma) \to B) \Rightarrow \Xi$ to $(A \Rightarrow \Sigma) \to (B \Rightarrow \Xi)$ is needed to type abstraction and application. This latter conversion rule has no computational content.

In order to prove subject reduction of the type system in Figure 12 we state two auxiliary lemmas (Generation and Substitution Lemma).

LEMMA 13 (Generation Lemma). *1.* $\Gamma, x : A_\Sigma \vdash_\Xi x : B; \Delta$ *implies* $\Xi \equiv \Sigma$ *and* $B \equiv A$.

2. $\Gamma \vdash_\Xi \lambda x.M : C; \Delta$ *implies* $C \equiv A_\Sigma \to B$ *and* $\Gamma, x : A_\Sigma \vdash_\Xi M : B; \Delta$.

3. $\Gamma \vdash_\Xi MN : B; \Delta$ *implies* $\Gamma \vdash_\Xi M : A_\Sigma \to B; \Delta$ *and* $\Gamma \vdash_\Sigma N : A; \Delta$ *for some* A *and* Σ.

4. $\Gamma \vdash_\Sigma \mu\alpha.c : A; \Delta$ *implies* $\Gamma \vdash_\Sigma c : \perp\!\!\!\perp; \Delta, \alpha : A$.

5. $\Gamma \vdash_\Sigma [\alpha]M : A; \Delta$ *implies* $\Delta \equiv \Delta', \alpha : A$ *and* $\Gamma \vdash_\Sigma M : \perp\!\!\!\perp; \Delta$.

6. $\Gamma \vdash_\Sigma \mu\widehat{\mathsf{tp}}.c : A; \Delta$ *implies* $\Gamma \vdash_{A\cdot\Sigma} c : \perp\!\!\!\perp; \Delta$.

7. $\Gamma \vdash_\Xi [\widehat{\mathsf{tp}}]c : \perp\!\!\!\perp; \Delta$ *implies* $\Xi \equiv A \cdot \Sigma$ *and* $\Gamma \vdash_\Sigma M : A; \Delta$.

LEMMA 14 (Substitution lemma). *1. Let* $\Gamma, x : A_\Sigma \vdash_\Xi M : B; \Delta$ *and* $\Gamma \vdash_\Sigma N : A; \Delta$. *Then* $\Gamma \vdash_\Xi M[N/x] : B; \Delta$.

2. Let $\Gamma \vdash_\Xi c : \perp\!\!\!\perp; \Delta, \alpha : A_\Sigma \to B$ *and* $\Gamma \vdash_\Sigma N : A; \Delta$ *and let* β *be a fresh variable. Then* $\Gamma \vdash_\Xi c[[\beta](\Box\, N)/\alpha] : \perp\!\!\!\perp; \Delta, \beta : B$.

3. Let $\Gamma \vdash_\Sigma c : \perp\!\!\!\perp; \Delta, \alpha : A, \beta : A$. *Then* $\Gamma \vdash_\Sigma c[\beta/\alpha] : \perp\!\!\!\perp; \Delta, \beta : A$.

Subject reduction follows directly.

PROPOSITION 15 (Subject reduction).

(i) If $\Gamma \vdash_\Sigma M : A; \Delta$ *and* $M \to N$, *then* $\Gamma \vdash_\Sigma N : A; \Delta$.

(ii) If $\Gamma \vdash_\Sigma c : \perp\!\!\!\perp; \Delta$ *and* $c \to c'$, *then* $\Gamma \vdash_\Sigma c' : \perp\!\!\!\perp; \Delta$.

4.5 Continuation-passing-style semantics

De Groote (1994) defined a CPS transformation to λ-calculus for $\Lambda\mu$. We give here an alternative CPS transformation that is based on a call-by-name CPS translation to λ-calculus with pairs (Lafont, Reus, and Streicher 1993). The λ-calculus with pairs is defined by the syntax

$$
M \ ::= \ x \mid \lambda x.M \mid M\ M \mid (M, M) \mid \mathbf{let}\ (y, x) = M\ \mathbf{in}\ M
$$

and we use $\lambda(x, y).t$ as an abbreviation for $\lambda z.\mathbf{let}\ (x, y) = z\ \mathbf{in}\ t$ for z fresh. In addition to (β) and (η), the calculus comes with the following reduction rules:

$$
\begin{array}{lll}
\wedge : & \mathbf{let}\ (x, y) = (M, N)\ \mathbf{in}\ M' & \to\ M'\ [N/y; M/x] \\
\wedge_{\mathit{lift}} : & F[\mathbf{let}\ (x, y) = M\ \mathbf{in}\ N] & \to\ \mathbf{let}\ (x, y) = M\ \mathbf{in}\ F[N] \\
\eta_\wedge : & \mathbf{let}\ (x, y) = M\ \mathbf{in}\ (x, y) & \to\ M
\end{array}
$$

for $F ::= \Box\ N \mid \mathbf{let}\ (x, y) = \Box\ \mathbf{in}\ M$.

We assume to have an injection k_α from continuation variables to term variables. The CPS transformation is shown in Figure 13. To the exception of some uses of η-conversion, it differs from de Groote's transformation on $\Lambda\mu$ only in the application and abstraction cases.

$$
\begin{array}{lcl}
x^* & \triangleq & x \\
(\lambda x.M)^* & \triangleq & \lambda(x, k).M^*\, k \\
(M\, N)^* & \triangleq & \lambda k.M^*\, (N^*, k) \\
(\mu\alpha.c)^* & \triangleq & \lambda k_\alpha.c^* \\
([\alpha]M)^* & \triangleq & M^*\, k_\alpha \\
(\mu\widehat{\mathsf{tp}}.c)^* & \triangleq & c^* \\
([\widehat{\mathsf{tp}}]M)^* & \triangleq & M^*
\end{array}
$$

Figure 13. Call-by-name CPS translation of $\lambda\mu\widehat{\mathsf{tp}}$

The CPS transformation is compatible with the type system. Indeed, if we define the following transformation on types:

$$
\begin{array}{lcl}
X^- & \triangleq & \neg X \\
(A_\Sigma \to B)^- & \triangleq & {A_\Sigma}^+ \wedge B^- \\
\\
{A_\Sigma}^+ & \triangleq & A^- \to \Sigma^+ \\
\\
\perp^+ & \triangleq & \perp \\
(A \cdot \Sigma)^+ & \triangleq & A^- \to \Sigma^+ \\
\\
(\emptyset)^+ & \triangleq & \emptyset \\
(\Gamma, x : A_\Sigma)^+ & \triangleq & \Gamma^+, x : {A_\Sigma}^+ \\
(\emptyset)^- & \triangleq & \emptyset \\
(\Delta, \alpha : A)^- & \triangleq & \Delta^-, \alpha : A^- \\
\\
(\Gamma \vdash_\Sigma M : A; \Delta)^+ & \triangleq & \Gamma^+, \Delta^- \vdash M^* : A^- \to \Sigma^+ \\
(\Gamma \vdash_\Sigma c : \perp\!\!\!\perp; \Delta)^+ & \triangleq & \Gamma^+, \Delta^- \vdash c^* : \Sigma^+
\end{array}
$$

then, we get the following compatibility result:

PROPOSITION 16.

(i) If $\Gamma \vdash_\Sigma M : A; \Delta$ *then* $(\Gamma \vdash_\Sigma M : A; \Delta)^+$.
(ii) If $\Gamma \vdash_\Sigma c : \perp\!\!\!\perp; \Delta$ *then* $(\Gamma \vdash_\Sigma c : \perp\!\!\!\perp; \Delta)^+$.

Unfortunately, the CPS above does not simulate the reduction. As it is common, we would have needed a CPS that takes care of administrative redex to get a simulation result. Still, the CPS above is compatible with equality in the λ-calculus with pairs:

PROPOSITION 17. *If* $M \to N$ *then* $M^* = N^*$.

We can also state a completeness result (this is an adaptation of standard proofs, see e.g. de Groote 1994; Fujita 2003):

PROPOSITION 18. *If* $M^* =_{\beta\eta\wedge\wedge_{lift}\eta_\wedge} N^*$ *then* $M = N$.

REMARK: In the very same way as for call-by-value $\lambda\mu\widehat{\mathtt{tp}}$, the call-by-name CPS translation can be factorised as the composition first of a state-passing-style transformation to call-by-name $\lambda\mu$ extended with an asymmetric disjunction (because the type effects in call-by-name $\lambda\mu\widehat{\mathtt{tp}}$ "naturally" take the form of an asymmetric disjunction; for asymmetric disjunction, see Pym and Ritter 2001), then of a call-by-name CPS translation to the λ-calculus with pairs.

4.6 Operational semantics

We first give the operational semantics of call-by-name $\lambda\mu\widehat{\mathtt{tp}}$ as a set of reduction rules applicable to the term as a whole. This kind of operational semantics in "natural" style is defined on commands by the following rules:

$$\begin{array}{llll}
\beta : & D_n[[\alpha]E_n[(\lambda x.M)\,N]] & \mapsto & D_n[[\alpha]E_n[M[N/x]]] \\
\mu_n : & D_n[[\alpha]E_n[\mu\beta.c]] & \mapsto & D_n[c[[\alpha]E_n/\beta]] \\
\eta^n_{\widehat{\mathtt{tp}}} : & D_n[[\alpha]E_n[\mu\widehat{\mathtt{tp}}.[\widehat{\mathtt{tp}}]M]] & \mapsto & D_n[[\alpha]E_n[M]]
\end{array}$$

As in the call-by-value case, $\mapsto$ is included in $\twoheadrightarrow$ of which it constitutes on commands a level of abstraction. We say that c is a *weak-head normal* command if for no c', $c \mapsto c'$. Weak-head normal commands are either of the form $[\widehat{\mathtt{tp}}]M$, or of the form $D_n[[\alpha]E_n[x]]$ or of the form $D_n[[\alpha]\lambda x.M]$.

We then present the operational semantics by means of a call-by-name abstract machine. The language of the abstract machine for call-by-name is shown in Figure 14 and the reduction steps are given in Figure 15. As for the call-by-value machine in Section 3, the language of the machine is an extension with explicit environments of the language of $\lambda\mu\widehat{\mathtt{tp}}$. To initiate the computation, we need an extra constant of evaluation context that we write ϵ.

As in the call-by-value machine, the evaluation rules are split into three categories. However, the control is first owned by the evaluation context, so that the "logical" steps are controlled not by the value but by the linear evaluation context. Final result reconstruction in terminal states uses almost the same operations as for the call-by-value machine.

$$\begin{array}{lcll}
[\alpha[e]]^\dagger & \triangleq & L^\dagger & \text{if } e(\alpha) = L \\
[\alpha[e]]^\dagger & \triangleq & [\alpha](\Box) & \text{otherwise} \\
(M[e]\cdot L)^\dagger & \triangleq & L^\dagger[\Box\ M[e]^\dagger] & \\
 & & & \\
[\,]^\dagger & \triangleq & \Box & \\
[\widehat{\mathtt{tp}} = L; S]^\dagger & \triangleq & S^\dagger[L^\dagger[\mu\widehat{\mathtt{tp}}.\Box]] & \\
M[x = N[e']; e]^\dagger & \triangleq & M[N[e']^\dagger/x][e]^\dagger & \\
M[\alpha = L; e]^\dagger & \triangleq & M[L^\dagger/\alpha][e]^\dagger &
\end{array}$$

PROPOSITION 19. *If c is a weak-head normal form, then,* $[\epsilon]M \mapsto\!\!\!\!\twoheadrightarrow c$ *iff the evaluation starting from* $\langle M\ [\,]\ ||\ \epsilon[\,]\rangle_{eval}\ [\,]$ *stops with result c.*

4.7 An example

How does call-by-name $\lambda\mu\widehat{\mathtt{tp}}$ behave on standard examples that uses delimited control? We consider the example of list traversal that Biernacki and Danvy (2005) used to emphasise the differences between operator $\mathcal{F}$ (Felleisen 1988) and *shift*. We extend $\lambda\mu\widehat{\mathtt{tp}}$ with a fixpoint operator, list constructors and a list destructor:

$$\begin{array}{lcl}
M, N & ::= & \ldots \mid \nu_x.M \mid [] \mid M{::}N \\
 & \mid & \mathtt{if}\ M\ \mathtt{is}\ x{::}y\ \mathtt{then}\ M\ \mathtt{else}\ M
\end{array}$$

and we extend call-by-name reduction with the rules

$$\begin{array}{lcl}
\nu_x.M & \to & M[\nu_x.M/x] \\
\mathtt{if}\ []\ \mathtt{is}\ x{::}y\ \mathtt{then}\ M_2\ \mathtt{else}\ M_1 & \to & M_1 \\
\mathtt{if}\ M{::}N\ \mathtt{is}\ x{::}y\ \mathtt{then}\ M_2\ \mathtt{else}\ M_1 & \to & M_2[M/x][N/y] \\
\mathtt{if}\ \mu\alpha.c\ \mathtt{is}\ x{::}y\ \mathtt{then}\ M_2\ \mathtt{else}\ M_1 & \to & \\
\quad \mu\alpha.c[[\alpha]\,(\mathtt{if}\ \Box\ \mathtt{is}\ x{::}y\ \mathtt{then}\ M_2\ \mathtt{else}\ M_1)/\alpha] & &
\end{array}$$

In informal ML syntax, the example is the following

```
let traverse l =
  let rec visit l = match l with
  | [] -> []
  | a::l' -> visit (shift (fun k -> a :: k l'))
  in reset (visit l)
in traverse [1;2;3]
```

Translated into $\Lambda\mu$, it gives

$$v\ (n_1{::}n_2{::}n_3{::}[])$$

where v is $\nu_f.(\lambda l.\mathtt{if}\ l\ \mathtt{is}\ a{::}l'\ \mathtt{then}\ f\ (\mu\alpha.a{::}[\alpha]l')\ \mathtt{else}\ [])$. Translated into $\lambda\mu\widehat{\mathtt{tp}}$, v is

$$\nu_f.(\lambda l.\mathtt{if}\ l\ \mathtt{is}\ a{::}l'\ \mathtt{then}\ f\ (\mu\alpha.[\widehat{\mathtt{tp}}]a{::}\mu\widehat{\mathtt{tp}}.[\alpha]l')\ \mathtt{else}\ [])\,.$$

Let ϵ be an arbitrary continuation distinct from $\widehat{\mathtt{tp}}$. We write l_i for $n_i{::}\ldots{::}n_3{::}[]$. We list the steps of the reduction of $[\epsilon](v\ l_1)$:

$$\begin{array}{l}
[\epsilon]v\ l_1 \\
\to\ [\epsilon](\lambda l.\mathtt{if}\ l\ \mathtt{is}\ a{::}l'\ \mathtt{then}\ v\ (\mu\alpha.a{::}[\alpha]l')\ \mathtt{else}\ [])\,l_1 \\
\to\ [\epsilon]\mathtt{if}\ l_1\ \mathtt{is}\ a{::}l'\ \mathtt{then}\ v\ (\mu\alpha.a{::}[\alpha]l')\ \mathtt{else}\ []) \\
\to\ [\epsilon]v\ (\mu\alpha.n_1{::}[\alpha]l_2) \\
\twoheadrightarrow \mathtt{if}\ (\mu\alpha.n_1{::}[\alpha]l_2)\ \mathtt{is}\ a{::}l'\ \mathtt{then}\ v\ (\mu\alpha.a{::}[\alpha]l')\ \mathtt{else}\ [] \\
\to\ [\epsilon]\mu\alpha.n_1{::}[\alpha](\mathtt{if}\ l_2\ \mathtt{is}\ a{::}l'\ \mathtt{then}\ v\ (\mu\alpha.a{::}[\alpha]l')\ \mathtt{else}\ []) \\
\to\ n_1{::}[\epsilon](\mathtt{if}\ l_2\ \mathtt{is}\ a{::}l'\ \mathtt{then}\ v\ (\mu\alpha.a{::}[\alpha]l')\ \mathtt{else}\ []) \\
\to\ n_1{::}[\epsilon](v\ (\mu\alpha.n_2{::}[\alpha]l_3)) \\
\twoheadrightarrow n_1{::}[\epsilon](\mu\alpha.n_2{::}[\alpha](v\ l_3)) \\
\twoheadrightarrow n_1{::}n_2{::}[\epsilon](v\ l_3) \\
\twoheadrightarrow n_1{::}n_2{::}[\epsilon](\mu\alpha.n_3{::}[\alpha](v\ [])) \\
\twoheadrightarrow n_1{::}n_2{::}n_3{::}[\epsilon](v\ []) \\
\twoheadrightarrow n_1{::}n_2{::}n_3{::}[\epsilon][]
\end{array}$$

Otherwise said, the list traversal program copies its argument and shifts its continuation to the tail of the list.

5. Discussion on a General Framework for Calculi of Delimited Continuations

We review below two variants of the original calculus with *shift* and *reset*. Together with $\Lambda\mu$, we then obtain four calculi of delimited continuations. We show how these four calculi are related.

Lazy reset A variant of call-by-value $\lambda\mu\widehat{\mathtt{tp}}$ can be obtained by considering that terms of the form $\mu\widehat{\mathtt{tp}}.c$ are values. In this case, one obtains a calculus equivalent to the λ-calculus with *shift* and *lazy reset*, a calculus for which Sabry gave an axiomatisation complete with respect to its CPS semantics (Sabry 1996).

$$
\begin{array}{lll}
K & ::= & \widehat{\mathsf{tp}}[e] \mid L \\
L & ::= & \alpha[e] \mid M[e] \cdot L \\
[S] & ::= & [\,] \mid [\widehat{\mathsf{tp}} = L; S] \\
[e] & ::= & [\,] \mid [x = M[e]; e] \mid [\alpha = L; e] \\
s & ::= & \langle M[e] \mid\mid K\rangle_{\mathit{cont}}\,[S] \mid \langle M\,[e] \mid\mid L\rangle_{\mathit{eval}}\,[S] \mid \langle \lambda x.M[e] \mid\mid L\rangle_{\mathit{logic}}\,[S]
\end{array}
\quad
\begin{array}{l}
\text{(evaluation contexts)} \\
\text{(linear evaluation contexts)} \\
\text{(dynamic environment)} \\
\text{(environments)} \\
\text{(states)}
\end{array}
$$

Figure 14. Specific components of the abstract machine for call-by-name $\lambda\mu\widehat{\mathsf{tp}}$-calculus

Control owned by the evaluation context

$$
\begin{array}{llllllll}
\langle M[e] & \mid\mid L & \rangle_{\mathit{cont}} & [S] & \rightarrow & \langle M\ [e] & \mid\mid L & \rangle_{\mathit{eval}}\ [S] \\
\langle M[e] & \mid\mid \widehat{\mathsf{tp}}[e'] & \rangle_{\mathit{cont}} & [\widehat{\mathsf{tp}} = L; S] & \rightarrow & \langle M\ [e] & \mid\mid L & \rangle_{\mathit{eval}}\ [S] \\
\langle M[e] & \mid\mid \widehat{\mathsf{tp}}[e'] & \rangle_{\mathit{cont}} & [\,] & \rightarrow & \text{stop on } [\widehat{\mathsf{tp}}]M[e]^{\dagger} & &
\end{array}
$$

Control owned by the term

$$
\begin{array}{lllllllll}
\langle x & [e] \mid\mid L & \rangle_{\mathit{eval}} & [S] & \rightarrow & \langle M\ [e'] & \mid\mid L & \rangle_{\mathit{eval}}\ [S] & \text{if } e(x) = M[e'] \\
\langle \lambda x.M & [e] \mid\mid L & \rangle_{\mathit{eval}} & [S] & \rightarrow & \langle \lambda x.M[e] & \mid\mid L & \rangle_{\mathit{logic}}\ [S] & \\
\langle M\,N & [e] \mid\mid L & \rangle_{\mathit{eval}} & [S] & \rightarrow & \langle M\ [e] & \mid\mid N[e] \cdot L & \rangle_{\mathit{eval}}\ [S] & \\
\langle \mu\alpha.[q]M & [e] \mid\mid L & \rangle_{\mathit{eval}} & [S] & \rightarrow & \langle M[\alpha = L; e] & \mid\mid q[\alpha = L; e] & \rangle_{\mathit{cont}}\ [S] & \\
\langle \mu\widehat{\mathsf{tp}}.[q]M & [e] \mid\mid L & \rangle_{\mathit{eval}} & [S] & \rightarrow & \langle M[e] & \mid\mid q[e] & \rangle_{\mathit{cont}}\ [\widehat{\mathsf{tp}} = L; S] & \\
\langle x & [e] \mid\mid L & \rangle_{\mathit{eval}} & [S] & \rightarrow & \text{stop on } S^{\dagger}[L^{\dagger}[x]] & & & \text{if } x \text{ not bound in } e
\end{array}
$$

Control owned by the linear evaluation context

$$
\begin{array}{lllllllll}
\langle \lambda x.M[e] & \mid\mid M'[e'] \cdot L & \rangle_{\mathit{logic}} & [S] & \rightarrow & \langle M\ [x = M'[e']; e] & \mid\mid L & \rangle_{\mathit{eval}}\ [S] & \\
\langle \lambda x.M[e] & \mid\mid \alpha[e'] & \rangle_{\mathit{logic}} & [S] & \rightarrow & \langle \lambda x.M[e] & \mid\mid L & \rangle_{\mathit{logic}}\ [S] & \text{if } e'(\alpha) = L \\
\langle \lambda x.M[e] & \mid\mid \alpha[e'] & \rangle_{\mathit{logic}} & [S] & \rightarrow & \text{stop on } S^{\dagger}[[\alpha](\lambda x.M[e]^{\dagger})] & & & \text{if } \alpha \text{ not bound in } e'
\end{array}
$$

To evaluate M, we need a linear toplevel free variables distinct from $\widehat{\mathsf{tp}}$ (which is not linear). Let ϵ be this variable. Then, the machine starts with the following initial state:

$$\langle M\,[\,] \mid\mid \epsilon[\,]\rangle_{\mathit{eval}}\,[\,]$$

Figure 15. Abstract machine for call-by-name $\lambda\mu\widehat{\mathsf{tp}}$-calculus

Call-by-name shift/reset with "lazy" toplevel continuation The first author once asked Olivier Danvy: What would be a "canonical" call-by-name variant of the *shift/reset* calculus? O. Danvy answered by an abstract machine that modifies the pure λ-calculus part of the machine for *shift*/*reset* in Biernacka et al. (2003) so that it behaves in call-by-name discipline. Expressed in the language of $\lambda\mu\widehat{\mathsf{tp}}$, the resulting calculus differs from the call-by-name variant of $\lambda\mu\widehat{\mathsf{tp}}$ studied in the paper in that the rules (μ_{var}) and $(\eta_{\widehat{\mathsf{tp}}})$ are now those of call-by-value $\lambda\mu\widehat{\mathsf{tp}}$:

$$
\begin{array}{llll}
\mu_{var}: & [q]\mu\alpha.c & \rightarrow & c[q/\alpha] \\
\eta_{\widehat{\mathsf{tp}}}: & \mu\widehat{\mathsf{tp}}.[\widehat{\mathsf{tp}}]\,V & \rightarrow & V
\end{array}
$$

Otherwise said, in this "lazy" call-by-name variant of $\lambda\mu\widehat{\mathsf{tp}}$, the toplevel continuation behaves as a regular linear evaluation context and it is captured by $\mu\alpha.c$ as the regular pieces of linear evaluation contexts $\square\ M$, $V\ \square$ and $[\beta]\ \square$ are.

The four calculi of delimited continuations The four calculi of delimited continuations are classified in Figure 16.

Choosing between call-by-name and call-by-value amounts to decide the *fundamental dilemma of computation* (as emphasised, e.g., in Curien and Herbelin 2000). Choosing call-by-value requires to restrict β-reduction into β_v-reduction and to add a rule (μ'_{app}) for incremental substitution of the new kind of context $(\lambda x.M)\,\square$.

In each variant, a subsidiary choice has to be made to decide if $\mu\widehat{\mathsf{tp}}$ is a value or not and if $\widehat{\mathsf{tp}}$ behaves like a linear continuation variable or not.

In call-by-value, the extra critical pair is $(\lambda x.t)\,(\mu\widehat{\mathsf{tp}}.c)$. If $\mu\widehat{\mathsf{tp}}.c$ is considered as non evaluated, the call-by-value discipline expects that priority is given to it and one obtains the original *shift* and *reset* calculus from Danvy and Filinski. If otherwise $\mu\widehat{\mathsf{tp}}.c$ is considered as evaluated, it yields its priority to its evaluation context, i.e. to the function, and β is applicable. One then obtains the calculus with *lazy reset* that was studied in Sabry (1996).

In call-by-name, the extra critical pair is $[\widehat{\mathsf{tp}}]\mu\alpha.c$. If priority is given to the evaluation context, i.e. $\widehat{\mathsf{tp}}$, one has first to know to what it is bound before to continue the computation. One then obtains the semantics of $\Lambda\mu$. If otherwise $\widehat{\mathsf{tp}}$ is considered as a linear continuation variable, it yields the priority to its argument and its capture is made possible. One then obtains Danvy's call-by-name variant of the *shift* and *reset* calculus.

6. Conclusions

Summary

We showed that de Groote variant of $\lambda\mu$-calculus, here called $\Lambda\mu$ after Saurin, while apparently similar to Parigot's $\lambda\mu$, can be interpreted as an extension of $\lambda\mu$ with call-by-name delimited control. Especially, we showed the following points:

Fundamental critical pair of computation
$(\lambda x.t)\ (\mu\alpha.c)$

$\swarrow$ (CBV) $\quad$ (CBN) $\searrow$

$(\beta_v) + (\mu_{app}) + (\mu'_{app}) + (\mu_{var}) + (\eta_{\widehat{\mathtt{tp}}})$ $\qquad$ $(\beta) + (\mu_{app}) + (\mu^n_{var}) + (\eta_{\widehat{\mathtt{tp}}})$

subsidiary choice
$(\lambda x.t)\ (\mu\widehat{\mathtt{tp}}.c)$

($\mu\widehat{\mathtt{tp}}$ value) $\swarrow$
shift/lazy reset
(Sabry)
CPS completion (Sabry)

$\searrow$ ($\mu\widehat{\mathtt{tp}}$ not value)
shift/reset
(Danvy-Filinski)
CPS completion (Kameyama-Hasegawa)
typed "domain"-completion (Sitaram-Felleisen)

subsidiary choice
$[\widehat{\mathtt{tp}}]\mu\alpha.c$

(μ^n_{var} moved to μ_{var}) $\swarrow$
CBN *shift/reset*
(Danvy)

$\searrow$ ($\eta_{\widehat{\mathtt{tp}}}$ moved to $\eta^n_{\widehat{\mathtt{tp}}}$)
$\Lambda\mu$
(de-Groote–Saurin)
Böhm-completion (Saurin)

Figure 16. Calculi of delimited continuations - a classification

- $\Lambda\mu$ can be interpreted as a call-by-name variant of Ariola et al's extension of call-by-value $\lambda\mu$ with delimited control, namely call-by-value $\lambda\mu\widehat{\mathtt{tp}}$.
- The abstract machine for $\Lambda\mu$ relies on a global stack for the dynamic continuation as the abstract machine for call-by-value $\lambda\mu\widehat{\mathtt{tp}}$ does.
- There is a system of simple types with effects for $\Lambda\mu$ for which subject reduction holds.

The $\Lambda\mu$ is a surprising calculus. On one side, its syntax and CPS semantics are very simple, and in particular simpler than the syntax and CPS semantics of call-by-value calculi of delimited continuations. On the other side, its "canonical" system of types and its operational semantics keep the complexity of a calculus of delimited control. The absence of an explicit control delimiter in $\Lambda\mu$ is at first glance surprising, but if we admit that the definition of $\langle M\rangle$ is $\mu\widehat{\mathtt{tp}}.[\widehat{\mathtt{tp}}]M$ as it is in call-by-value $\lambda\mu\widehat{\mathtt{tp}}$, then it is normal that no explicit $\langle M\rangle$ is needed in $\Lambda\mu$ since it collapses in call-by-name $\lambda\mu\widehat{\mathtt{tp}}$ to an identity operator. Another lesson is that the μ operator of $\Lambda\mu$ is indeed a *shift* operator[6].

One could ask whether the syntax of $\Lambda\mu$ can be used for call-by-value delimited control. The answer is yes if one adds an explicit $\langle M\rangle$. Indeed, in $\Lambda\mu$ extended with $\langle M\rangle$, the four combinations $\mu\alpha.[\beta]M$, $\mu\alpha.[\widehat{\mathtt{tp}}]M$, $\mu\widehat{\mathtt{tp}}.[\alpha]M$ and $\mu\widehat{\mathtt{tp}}.[\widehat{\mathtt{tp}}]M$ are equivalently expressible in $\Lambda\mu$ by $\mu\alpha.[\beta]M$, $\mu\alpha.M$, $[\alpha]M$ and $\langle M\rangle$ respectively.

Section 5 showed that $\Lambda\mu$ is not the only call-by-name delimited control. Further investigations into the four different calculi need to be done to better understand the relative strengths of each of the calculi.

The separability property in classical logic

The $\overline{\lambda}\mu\widetilde{\mu}$-calculus is the calculus of choice to study the kind of duality given in Figure 16. Uniformly investigating the completeness properties of the four calculi and completing the picture in the framework of $\overline{\lambda}\mu\widetilde{\mu}$-calculus would be interesting. Up to our knowledge, there are no results on Böhm's separability property in other proof calculi for classical logic. We believe that the separability property for call-by-name $\lambda\mu\widehat{\mathtt{tp}}$ would directly transfer to call-by-name untyped $\overline{\lambda}\mu\widetilde{\mu}$-calculus but Böhm's separability property in the untyped call-by-value and in the typed versions of $\overline{\lambda}\mu\widetilde{\mu}$-calculus are open problems. The question of separability in the Dual Calculus Wadler (2003) is a topic for future research, as well.

[6] In passing, this suggests that de Groote's use of $\Lambda\mu$ for representing quantifier scope in linguistic is not so far from the *shift/reset*-based approach of quantifier scope by Barker and Shan.

An other question is also the investigation of Böhm's theorem in the simply typed fragments of the four calculi (see e.g. Böhm's theorem in the simply typed λ-calculus by Došen and Petrić (2001), Statman (1982), Simpson (1995), Joly (2000)).

Finally, how far the study of Böhm's theorem in call-by-value calculus with control can help for investigating separation in Moggi's extension of Plotkin's λ_v (see Paolini 2001).

Acknowledgements We wish to thank Olivier Danvy and Alexis Saurin for fruitful discussions we had during the work on this paper.

References

Zena M. Ariola and Hugo Herbelin. Control reduction theories: the benefit of structural substitution. *J. Funct. Program.*, 2007. Includes a Historical Note by Matthias Felleisen. To appear.

Zena M. Ariola, Hugo Herbelin, and Amr Sabry. A type-theoretic foundation of delimited continuations. *Higher Order and Symbolic Computation*, 2007. To appear.

Kensuke Baba, Sachio Hirokawa, and Ken-etsu Fujita. Parallel reduction in type free $\lambda\mu$-calculus. *Electronic Notes in Theoretical Computer Science*, 42:52–66, 2001.

Małgorzata Biernacka, Dariusz Biernacki, and Olivier Danvy. An operational foundation for delimited continuations. Technical Report 03-41, BRICS, University of Aarhus, Denmark, 2003.

Dariusz Biernacki and Olivier Danvy. On the dynamic extent of delimited continuations. Technical Report 05-2, BRICS, University of Aarhus, Denmark, 2005.

Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming, ICFP 2000, Montreal, Canada, September 18-21, 2000*, SIGPLAN Notices 35(9), pages 233–243. ACM, 2000. ISBN 1-58113-202-6.

Vincent Danos, Hugo Herbelin, and Laurent Regnier. Game semantics & abstract machines. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science (LICS '96)*, pages 394–405. IEEE Computer Society Press, 1996.

Olivier Danvy and Andrzej Filinski. A functional abstraction of typed contexts. Technical Report 89/12, DIKU, University of Copenhagen, Copenhagen, Denmark, August 1989.

René David and Walter Py. Lambda-mu-calculus and Böhm's theorem. *J. Symb. Log.*, 66(1):407–413, 2001.

Philippe de Groote. A CPS-translation of the $\lambda\mu$-calculus. In S. Tison, editor, *Proceedings of the Colloquium on Trees in Algebra and Programming, CAAP'94, Edinburgh, U.K., April 11-13, 1994*, volume 787 of *Lecture Notes in Computer Science*, pages 85–99. Springer-Verlag, 1994. ISBN 3-540-57879-X.

Philippe de Groote. An environment machine for the lambda-mu-calculus. *Mathematical Structures in Computer Science*, 8(6):637–669, 1998.

Kosta Došen and Zoran Petrić. The typed Böhm teorem. In *Böhm theorem: applications to Computer Science Theory - BOTH 2001 Crete, Greece*, volume 50(2) of *Electronic Notes in Theoretical Computer Science*, page 13, 2001.

Matthias Felleisen. The theory and practice of first-class prompts. In *Proceedings of the 15th ACM Symposium on Principles of Programming Languages (POPL '88)*, pages 180–190. ACM Press, New York, January 1988.

Matthias Felleisen and Daniel Friedman. Control operators, the secd machine, and the lambda-calculus. In *Formal description of programming concepts-III*, pages 193–217. North-Holland, 1986.

Matthias Felleisen, Daniel P. Friedman, Eugene Kohlbecker, and Bruce F. Duba. Reasoning with continuations. In *First Symposium on Logic and Computer Science*, pages 131–141, 1986.

Andrzej Filinski. Representing monads. In *Conf. Record 21st ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, POPL'94, Portland, OR, USA, 17-21 Jan. 1994*, pages 446–457. ACM Press, New York, 1994.

Michael J. Fischer. Lambda-calculus schemata. In *Proc. ACM Conference on Proving Assertions About Programs*, volume 7(1) of *SIGPLAN Notices*, pages 104–109. ACM Press, New York, 1972.

Ken-etsu Fujita. A sound and complete cps-translation for lambda-mu-calculus. In *TLCA*, pages 120–134, 2003.

Timothy G. Griffin. The formulae-as-types notion of control. In *Conf. Record 17th Annual ACM Symp. on Principles of Programming Languages, POPL '90, San Francisco, CA, USA, 17-19 Jan 1990*, pages 47–57. ACM Press, New York, 1990.

Thérèse Hardin, Luc Maranget, and Bruno Pagano. Functional back-ends within the lambda-sigma calculus. In *ICFP*, pages 25–33, 1996.

Hugo Herbelin. Explicit substitutions and reducibility. *Journal of Logic and Computation*, 11(3):431–451, 2001.

Hugo Herbelin. *C'est maintenant qu'on calcule: au cœur de la dualité*. Habilitation à diriger les recherches, Université Paris 11, December 2005.

Hugo Herbelin. *Séquents qu'on calcule: de l'interprétation du calcul des séquents comme calcul de λ-termes et comme calcul de stratégies gagnantes*. Thèse de doctorat, Université Paris 7, January 1995.

Hugo Herbelin. Games and weak-head reduction for classical PCF. In Philippe de Groote and J. Roger Hindley, editors, *Third International Conference on Typed Lambda Calculi and Applications, TLCA '97, Nancy, France, April 2-4, 1997, Proceedings*, volume 1210 of *Lecture Notes in Computer Science*, pages 214–230. Springer, 1997. ISBN 3-540-62688-3.

Gregory F. Johnson. Gl – a denotational testbed with continuations and partial continuations as first-class objects. In *SIGPLAN '87: Papers of the Symposium on Interpreters and interpretive techniques*, pages 165–176, New York, NY, USA, 1987. ACM Press. ISBN 0-89791-235-7. doi: http://doi.acm.org/10.1145/29650.29668.

Gregory F. Johnson and Dominic Duggan. Stores and partial continuations as first-class objects in a language and its environment. In *POPL*, pages 158–168, 1988.

Thierry Joly. *Codages, séparabilité et représentation de fonctions dans divers λ-calculs typés*. Phd thesis, Université Paris 7, January 2000.

Yukiyoshi Kameyama and Masahito Hasegawa. A sound and complete axiomatization of delimited continuations. In Colin Runciman and Olin Shivers, editors, *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming, ICFP 2003, Uppsala, Sweden, August 25-29, 2003*, volume 38(9) of *SIGPLAN Notices*, pages 177–188. ACM Press, New York, 2003.

Yves Lafont, Bernhard Reus, and Thomas Streicher. Continuations semantics or expressing implication by negation. Technical Report 9321, Ludwig-Maximilians-Universitt, Mnchen, 1993.

C.-H. Luke Ong. A semantic view of classical proofs: type-theoretic, categorical, denotational characterizations. In *Proceedings of 11th IEEE Annual Symposium on Logic in Computer Science*, pages 230–241. IEEE Computer Society Press, 1996.

C.-H. Luke Ong and Charles A. Stewart. A Curry-Howard foundation for functional computation with control. In *Proceedings of ACM SIGPLAN-SIGACT Symposium on Principle of Programming Languages, Paris, January 1997*, pages 215–227. ACM Press, 1997.

Luca Paolini. Call-by-Value separability and computability. In A. Restivo and S. Ronchi della Rocca, editors, *7th Italian Conference Theoretical Computer Science, ICTCS'05*, volume 2202 of *Lecture Notes in Computer Science*, pages 74–89. Springer-Verlag, 2001.

Michel Parigot. Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction. In *Logic Programming and Automated Reasoning: International Conference LPAR '92 Proceedings, St. Petersburg, Russia*, pages 190–201. Springer-Verlag, 1992.

David Pym and Eike Ritter. On the semantics of classical disjunction. *Journal of Pure and Applied Algebra*, 159:315–338, 2001.

Amr Sabry. Note on axiomatizing the semantics of control operators. Technical Report CIS-TR-96-03, Dept of Information Science, Univ. of Oregon, 1996.

Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6(3-4): 289–360, 1993.

Alexis Saurin. Separation with streams in the $\lambda\mu$-calculus. In *Proceedings, 20th Annual IEEE Symposium on Logic in Computer Science (LICS '05)*, pages 356–365. IEEE Computer Society Press, 2005.

Alexis Saurin. Typing streams in the lambda-mu-calculus. In *14th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR '07)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2007. to appear.

Peter Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Mathematical Structures in Computer Science*, 11(2):207–260, 2001.

Chung-chieh Shan. Shift to control. In Olin Shivers and Oscar Waddell, editors, *Proceedings of the 5th workshop on Scheme and Functional Programming*, pages 99–107, 2004.

Alex K. Simpson. Categorical completeness results for simply typed lambda calculus. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Typed Lambda Calculus and Applications TLCA'95*, volume 902 of *Lecture Notes in Computer Science*, pages 414–427. Springer-Verlag, 1995.

Dorai Sitaram and Matthias Felleisen. Reasoning with continuations ii: full abstraction for models of control. In *LFP '90: Proceedings of the 1990 ACM conference on LISP and functional programming*, pages 161–175. ACM Press, 1990. ISBN 0-89791-368-X. doi: http://doi.acm.org/10.1145/91556.91626.

Richard Statman. Completeness, invariance and λ-definability. *J. Symb. Log.*, 47(1):17–26, 1982.

Philip Wadler. Call-by-value is dual to call-by-name. In Colin Runciman and Olin Shivers, editors, *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming, ICFP 2003, Uppsala, Sweden, August 25-29, 2003*, volume 38(9) of *SIGPLAN Notices*, pages 189–201. ACM, 2003. ISBN 1-58113-756-7.