PUBLICATIONS DE L'INSTITUT MATHÉMATIQUE
Nouvelle série, tome 92(106) (2012), 79–95 DOI: 10.2298/PIM1206079L

ON SEMANTICS OF A TERM CALCULUS FOR CLASSICAL LOGIC

Silvia Likavec and Pierre Lescanne

Communicated by Žarko Mijajlović

Abstract. The calculus of Curien and Herbelin was introduced to provide the Curry–Howard correspondence for classical logic. The terms of this calculus represent derivations in the sequent calculus proof system and reduction reflects the process of cut-elimination. We investigate some properties of two well-behaved subcalculi of untyped calculus of Curien and Herbelin, closed under the call-by-name and the call-by-value reduction, respectively. Continuation semantics is given using the category of negated domains and Moggi's Kleisli category over predomains for the continuation monad. Soundness theorems are given for both versions thus relating operational and denotational semantics. A thorough overview of the work on continuation semantics is given.

1. Introduction

It is well known that simply typed lambda calculus corresponds to intuitionistic logic through Curry–Howard correspondence [**15**]. Extending lambda calculus with control operators brings this correspondence to the realm of classical logic, as first showed by Griffin in [**13**]. Next cornerstone in the study of theories of control in programming languages was Parigot's $\lambda\mu$ calculus [**27**].

$\overline{\lambda}\mu\widetilde{\mu}$ calculus of Curien and Herbelin [**6**] is a system with a more fine-grained analysis of calculations within languages with control operators. Since it was introduced in [**6**], $\overline{\lambda}\mu\widetilde{\mu}$ calculus has had a strong influence on further understanding between calculi with control operators and classical logic (see [**2**, **3**, **39**, **40**]).

This work contributes to the better understanding of $\overline{\lambda}\mu\widetilde{\mu}$ calculus in two ways. We prove confluence and build denotational semantics for the *untyped* version of the calculus. Untyped $\overline{\lambda}\mu\widetilde{\mu}$ calculus is Turing-complete, hence a naive set-theoretic approach would not be enough. Since the calculus is not confluent, it is necessary to

2010 *Mathematics Subject Classification:* Primary 03B40, 03B70, 18C50; Secondary 68N18.
Key words and phrases: logic, lambda calculus, call-by-value, call-by-name, confluence, continuation semantics, control, monad, category theory.
Partially supported by the Ministry of Education and Science of Serbia, project ON174026.

consider separately the call-by-name and call-by-value disciplines. The semantics is defined using category of negated domains [**35**] and Kleisli category [**18**]. Soundness theorems are given for both, call-by-value and call-by-name subcalculi, thus relating operational and denotational semantics.

The paper is organized as follows. In Section 2 we recall the syntax and the reduction rules of $\overline{\lambda}\mu\widetilde{\mu}$ calculus, and its two well-behaved subcalculi $\overline{\lambda}\mu\widetilde{\mu}_T$ and $\overline{\lambda}\mu\widetilde{\mu}_Q$. In Section 3 we prove the confluence for $\overline{\lambda}\mu\widetilde{\mu}_T$ (the proof of confluence for $\overline{\lambda}\mu\widetilde{\mu}_Q$ being analogous). Section 4 presents an overview of the work done on continuation semantics, gives an account of negated domains and presents the basic notions of Kleisli triple and Kleisli category. We give the semantic interpretations of $\overline{\lambda}\mu\widetilde{\mu}_Q$ and $\overline{\lambda}\mu\widetilde{\mu}_T$ calculi in Subections 5.1 and 5.2. We conclude in Section 6.

2. Overview of $\overline{\lambda}\mu\widetilde{\mu}$ calculus

2.1. Intuition and syntax. The $\overline{\lambda}\mu\widetilde{\mu}$ calculus was introduced by Curien and Herbelin in [**6**], giving a Curry–Howard correspondence for classical logic. The terms of $\overline{\lambda}\mu\widetilde{\mu}$ represent derivations in the implicational fragment (hence without conjunction or disjunction) of the sequent calculus proof system LK and reduction reflects the process of cut-elimination.[1] The untyped version of the calculus can be seen as the foundation of a functional programming language with explicit notion of control and was further studied by Ghilezan and Lescanne in [**12**].

The syntax of $\overline{\lambda}\mu\widetilde{\mu}$ is given by the following, where v ranges over the set Caller of callers, e ranges over the set Callee of callees and c ranges over the set $\mathsf{Capsule}$ of capsules:

$$v ::= x \mid \lambda x.v \mid \mu\alpha.c \qquad e ::= \alpha \mid v \bullet e \mid \widetilde{\mu}x.c \qquad c ::= \langle v \parallel e \rangle$$

There are two kinds of variables in the calculus: the set Var_v, consisting of *caller variables* (denoted by Latin variables $x, y, \ldots$) and the set Var_e, consisting of *callee variables* (denoted by Greek variables $\alpha, \beta, \ldots$). The caller variables can be bound by λ abstraction or by μ abstraction, whereas the callee variables can be bound by $\widetilde{\mu}$ abstraction. The sets of free caller and callee variables, Fv_v and Fv_e, are defined as usual, respecting Barendregt's convention [**4**] that no variable can be both, bound and free, in the expression.

In [**6**], the basic constructs are called *commands, terms*, and *contexts*. In our opinion, meta-concepts like "terms" and "contexts" are going to be used naturally as in any other language, so it would be inappropriate to use them as concepts inside the language. In order to avoid confusion, in this work we use the following basic syntactic entities: the set Caller of callers, the set Callee of callees and the set $\mathsf{Capsule}$ of capsules, chosen by Ghilezan and Lescanne in [**12**].

Capsules are the place where callers and callees interact. A caller can either get the data from the callee, or it can ask the callee to replace one of its internal callee variables. A callee can ask a caller to replace one of its internal caller variables. The components can be nested and more processes can be active at the same time.

[1]Although, some cuts build normal forms in $\overline{\lambda}\mu\widetilde{\mu}$, as opposed to Lengrand's $\lambda\xi$ calculus [**20**], which is exactly LK (with implicit structural rules).

2.2. Reduction rules. There are only three rules that characterize the reduction in $\overline{\lambda}\mu\widetilde{\mu}$:

$$\begin{array}{rl} (\to') & \langle \lambda x.v_1 \parallel v_2 \bullet e\rangle \to \langle v_2 \parallel \widetilde{\mu}x.\langle v_1 \parallel e\rangle\rangle \\ (\mu) & \langle \mu\alpha.c \parallel e\rangle \to c[\alpha \leftarrow e] \\ (\widetilde{\mu}) & \langle v \parallel \widetilde{\mu}x.c\rangle \to c[x \leftarrow v] \end{array}$$

The above substitutions are defined as to avoid variable capture [**4**].

The calculus has a critical pair $\langle \mu\alpha.c_1 \parallel \widetilde{\mu}x.c_2\rangle$ where both (μ) and $(\widetilde{\mu})$ rule can be applied non-deterministically, producing two different results. For example,

$$\langle \mu\alpha.\langle y \parallel \beta\rangle \parallel \widetilde{\mu}x.\langle z \parallel \gamma\rangle\rangle \to_{\mu} \langle y \parallel \beta\rangle \quad \text{and} \quad \langle \mu\alpha.\langle y \parallel \beta\rangle \parallel \widetilde{\mu}x.\langle z \parallel \gamma\rangle\rangle \to_{\widetilde{\mu}} \langle z \parallel \gamma\rangle,$$

where α and β denote syntactically different callee variables.

Hence, the calculus is not confluent. But if the priority is given either to (μ) or to $(\widetilde{\mu})$ rule, we obtain two confluent subcalculi $\overline{\lambda}\mu\widetilde{\mu}_T$ and $\overline{\lambda}\mu\widetilde{\mu}_Q$ (we retain the original notation from [**6**]). We give the details in the next subsection.

2.3. Two confluent subcalculi. There are two possible reduction strategies in the calculus that depend on the orientation of the critical pair. If the priority is given to (μ) redexes, call-by-value reduction is obtained ($\overline{\lambda}\mu\widetilde{\mu}_Q$ calculus), whereas giving the priority to $(\widetilde{\mu})$ redexes, simulates call-by-name reduction ($\overline{\lambda}\mu\widetilde{\mu}_T$ calculus).

We first give the syntactic constructs of $\overline{\lambda}\mu\widetilde{\mu}_T$ and $\overline{\lambda}\mu\widetilde{\mu}_Q$, respectively:

$$\begin{array}{rl} \underline{\overline{\lambda}\mu\widetilde{\mu}_T} & \\ c ::= & \langle v \parallel e\rangle \\ v ::= & x \mid \lambda x.v \mid \mu\alpha.c \\ E ::= & \alpha \mid v \bullet E \\ e ::= & \widetilde{\mu}x.c \mid E \end{array} \qquad \begin{array}{rl} \underline{\overline{\lambda}\mu\widetilde{\mu}_Q} & \\ c ::= & \langle v \parallel e\rangle \\ V ::= & x \mid \lambda x.v \\ v ::= & \mu\alpha.c \mid V \\ e ::= & \alpha \mid \widetilde{\mu}x.c \mid V \bullet e \end{array}$$

In $\overline{\lambda}\mu\widetilde{\mu}_T$ the new syntactic subcategory E of callees, called *applicative contexts*, is introduced in order to model call-by-name reduction. In $\overline{\lambda}\mu\widetilde{\mu}_Q$, notice the presence of the new syntactic construct V that models the *values*.

The reduction rules for $\overline{\lambda}\mu\widetilde{\mu}_T$ and $\overline{\lambda}\mu\widetilde{\mu}_Q$ are the following:

$$\begin{array}{rl} \underline{\overline{\lambda}\mu\widetilde{\mu}_T} & \\ (\to) & \langle \lambda x.v_1 \parallel v_2 \bullet E\rangle \to \langle v_1[x \leftarrow v_2] \parallel E\rangle \\ (\mu) & \langle \mu\alpha.c \parallel E\rangle \to c[\alpha \leftarrow E] \\ (\widetilde{\mu}) & \langle v \parallel \widetilde{\mu}x.c\rangle \to c[x \leftarrow v] \\ & \\ \underline{\overline{\lambda}\mu\widetilde{\mu}_Q} & \\ (\to') & \langle \lambda x.v_1 \parallel V_2 \bullet e\rangle \to \langle V_2 \parallel \widetilde{\mu}x.\langle v_1 \parallel e\rangle\rangle \\ (\mu) & \langle \mu\alpha.c \parallel e\rangle \to c[\alpha \leftarrow e] \\ (\widetilde{\mu}) & \langle V \parallel \widetilde{\mu}x.c\rangle \to c[x \leftarrow V] \end{array}$$

Notice that our choice of rules does not violate the symmetry of Curien–Herbelin rules. In [**6**] only the rule $(\to')$ is considered for both subcalculi. In this paper we use $(\to)$ reduction rather than $(\to')$ reduction in the case of $\overline{\lambda}\mu\widetilde{\mu}_T$, since the application of the $(\to')$ rule will always be immediately followed by the application of the $(\widetilde{\mu})$ rule and that is exactly the rule $(\to)$. We think that our choice makes explicit the priorities of the rules in each subcalculus.

3. Confluence

Since in the next sections we work with two confluent subcalculi of $\overline{\lambda}\mu\widetilde{\mu}$ calculus, and confluence was never spelled out so far, we think it is in place to prove the confluence for each of them. We adopt the technique of parallel reduction given by Takahashi in [**38**]. This approach consists of simultaneously reducing all the redexes existing in a term.

We give the proof only for $\overline{\lambda}\mu\widetilde{\mu}_T$, since the proof for $\overline{\lambda}\mu\widetilde{\mu}_Q$ is obtained by a straightforward modification of the proof for $\overline{\lambda}\mu\widetilde{\mu}_T$. The complete proofs can be found in [**21**]. We denote the reduction defined by the three reduction rules for $\overline{\lambda}\mu\widetilde{\mu}_T$ by $\to_n$ and its reflexive, transitive and closure by congruence by $\twoheadrightarrow_n$.

First, we define the notion of parallel reduction $\Rightarrow_n$ for $\overline{\lambda}\mu\widetilde{\mu}_T$. We prove that $\twoheadrightarrow_n$ is reflexive and transitive closure of $\Rightarrow_n$ (Lemma 3.3), so in order to prove the confluence of $\twoheadrightarrow_n$, it is enough to prove the diamond property for $\Rightarrow_n$ (Theorem 3.2). The diamond property for $\Rightarrow_n$, follows from the stronger "Star property" for $\Rightarrow_n$ that we prove (Theorem 3.1).

3.1. Parallel reduction for $\overline{\lambda}\mu\widetilde{\mu}_T$ calculus.

DEFINITION 3.1 (Parallel reduction for $\overline{\lambda}\mu\widetilde{\mu}_T$ calculus). The parallel reduction, denoted by $\Rightarrow_n$ is defined inductively, as follows:

$$\frac{}{x \Rightarrow_n x}\ (g1_n) \qquad \frac{v \Rightarrow_n v'}{\lambda x.v \Rightarrow_n \lambda x.v'}\ (g2_n) \qquad \frac{c \Rightarrow_n c'}{\mu\alpha.c \Rightarrow_n \mu\alpha.c'}\ (g3_n)$$

$$\frac{}{\alpha \Rightarrow_n \alpha}\ (g4_n) \qquad \frac{v \Rightarrow_n v', E \Rightarrow_n E'}{v \bullet E \Rightarrow_n v' \bullet E'}\ (g5_n) \qquad \frac{c \Rightarrow_n c'}{\widetilde{\mu}x.c \Rightarrow_n \widetilde{\mu}x.c'}\ (g6_n)$$

$$\frac{v \Rightarrow_n v', e \Rightarrow_n e'}{\langle v \parallel e\rangle \Rightarrow_n \langle v' \parallel e'\rangle}\ (g7_n) \qquad \frac{v_1 \Rightarrow_n v_1', v_2 \Rightarrow_n v_2', E \Rightarrow_n E'}{\langle \lambda x.v_1 \parallel v_2 \bullet E\rangle \Rightarrow_n \langle v_1'[x \leftarrow v_2'] \parallel E'\rangle}\ (g8_n)$$

$$\frac{c \Rightarrow_n c', E \Rightarrow_n E'}{\langle \mu\alpha.c \parallel E\rangle \Rightarrow_n c'[\alpha \leftarrow E']}\ (g9_n) \qquad \frac{v \Rightarrow_n v', c \Rightarrow_n c'}{\langle v \parallel \widetilde{\mu}x.c\rangle \Rightarrow_n c'[x \leftarrow v']}\ (g10_n)$$

LEMMA 3.1. *For every term G, $G \Rightarrow_n G$.*

PROOF. Induction on the structure of G. The base cases are the rules $(g1_n)$ and $(g4_n)$ from Definition 3.1. For any other term of the calculus, we apply the induction hypothesis to the immediate subterms of G (rules $(g2_n)$, $(g3_n)$, $(g5_n)$–$(g7_n)$). $\square$

LEMMA 3.2 (Substitution lemma).

1. $G[x \leftarrow v_1][y \leftarrow v_2] = G[y \leftarrow v_2][x \leftarrow v_1[y \leftarrow v_2]]$, *for* $x \neq y$, $x \notin \mathrm{Fv}_v(v_2)$.
2. $G[x \leftarrow v][\alpha \leftarrow e] = G[\alpha \leftarrow e][x \leftarrow v[\alpha \leftarrow e]]$, *for* $x \notin \mathrm{Fv}_v(e)$.
3. $G[\alpha \leftarrow e][x \leftarrow v] = G[x \leftarrow v][\alpha \leftarrow e[x \leftarrow v]]$, *for* $\alpha \notin \mathrm{Fv}_e(v)$.
4. $G[\alpha \leftarrow e_1][\beta \leftarrow e_2] = G[\beta \leftarrow e_2][\alpha \leftarrow e_1[\beta \leftarrow e_2]]$, *for* $\alpha \neq \beta$, $\alpha \notin \mathrm{Fv}_e(e_2)$.

PROOF. Induction on the structure of G. It is enough to prove the first two statements for the caller variables, and the last two statements for the callee variables only, since all the other cases are either trivial or follow directly from the induction hypothesis. $\square$

Next, we give the definition of *contexts*, which are terms with the "hole" and are used in the proof of Lemma 3.3.

DEFINITION 3.2 (Contexts).

$$C ::= [\,] \mid \lambda x.C \mid \mu\alpha.C \mid v \bullet C \mid C \bullet E \mid \widetilde{\mu}x.C \mid \langle v \parallel C\rangle \mid \langle C \parallel e\rangle$$

With $C[G]$ we denote "filling the hole" of the context C with the term G (with possible variable capture).

LEMMA 3.3. 1. *If* $G \rightarrow_n G'$ *then* $G \Rightarrow_n G'$; 2. *If* $G \Rightarrow_n G'$ *then* $G \twoheadrightarrow_n G'$; 3. *If* $G \Rightarrow_n G'$ *and* $H \Rightarrow_n H'$, *then*

$$G[x \leftarrow H] \Rightarrow_n G'[x \leftarrow H'] \text{ and } G[\alpha \leftarrow H] \Rightarrow_n G'[\alpha \leftarrow H'].$$

PROOF. 1. Induction on the context of the redex. If $G \rightarrow_n G'$ then $G = \mathcal{C}[H]$, $G' = \mathcal{C}[H']$ and $H \rightarrow_n H'$. We just show two illustrative cases:

* $C = [\,]$. We proceed by induction on the definition of $H \rightarrow_n H'$. We have the following cases:
  - $H = \langle \mu\alpha.c \parallel E\rangle$ and $H' = c[\alpha \leftarrow E]$. Then $H \Rightarrow_n H'$ by $(g9_n)$, because $c \Rightarrow_n c$ and $E \Rightarrow_n E$ by Lemma 3.1.
  - Cases $H = \langle \lambda x.v_1 \parallel v_2 \bullet E\rangle$ and $H = \langle v \parallel \widetilde{\mu}x.c\rangle$ are treated similarly.
* $C = \widetilde{\mu}x.C'$. Then $G = \widetilde{\mu}x.C'[H]$ and $G' = \widetilde{\mu}x.C'[H']$. By the induction hypothesis, $C'[H] \Rightarrow_n C'[H']$, so by $(g3_n)$ of the Definition 3.1 we get $G \Rightarrow_n G'$.

2. Induction on the definition of $G \Rightarrow_n G'$. Since the proofs follow the same pattern in all the cases, we show just the case when

$$G = \langle \lambda x.v_1 \parallel v_2 \bullet E\rangle \Rightarrow_n \langle v_1'[x \leftarrow v_2'] \parallel E'\rangle = G'.$$

This follows directly from $v_1 \Rightarrow_n v_1'$, $v_2 \Rightarrow_n v_2'$ and $E \Rightarrow_n E'$. By the induction hypothesis, $v_i \twoheadrightarrow_n v_i'$, $i = 1, 2$ and $E \twoheadrightarrow_n E'$ so it follows that

$$G = \langle \lambda x.v_1 \parallel v_2 \bullet E\rangle \rightarrow_n \langle v_1[x \leftarrow v_2] \parallel E\rangle \twoheadrightarrow_n \langle v_1'[x \leftarrow v_2'] \parallel E'\rangle = G'.$$

3. Induction on the definition of $G \Rightarrow_n G'$. We only illustrate the proof of $G[x \leftarrow H] \Rightarrow_n G'[x \leftarrow H']$, since the proof of $G[\alpha \leftarrow H] \Rightarrow_n G'[\alpha \leftarrow H']$ follows the same pattern (using cases 3 and 4 of the Substitution lemma 3.2).

Let $G = \langle \lambda y.v_1 \parallel v_2 \bullet E\rangle \Rightarrow_n \langle v_1'[y \leftarrow v_2'] \parallel E'\rangle = G'$. This is a direct consequence of $v_1 \Rightarrow_n v_1'$, $v_2 \Rightarrow_n v_2'$, and $E \Rightarrow_n E'$. By the induction hypothesis it follows that $v_1[x \leftarrow H] \Rightarrow_n v_1'[x \leftarrow H']$, $v_2[x \leftarrow H] \Rightarrow_n v_2'[x \leftarrow H']$, and $E[x \leftarrow H] \Rightarrow_n E'[x \leftarrow H']$. Then, using Lemma 3.2(1) and $(g8_n)$ we derive

$$\begin{aligned}
G[x \leftarrow H] &= \langle \lambda y.v_1 \parallel v_2 \bullet E\rangle[x \leftarrow H] \\
&= \langle \lambda y.v_1[x \leftarrow H] \parallel v_2[x \leftarrow H] \bullet E[x \leftarrow H]\rangle \\
&\Rightarrow_n \langle v_1'[x \leftarrow H'][y \leftarrow v_2'[x \leftarrow H']] \parallel E'[x \leftarrow H']\rangle \\
&= \langle v_1'[y \leftarrow v_2'][x \leftarrow H'] \parallel E'[x \leftarrow H']\rangle = G'[x \leftarrow H'].
\end{aligned}$$

$\square$

From 1 and 2 we conclude that $\twoheadrightarrow_n$ is the reflexive and transitive closure of $\Rightarrow_n$.

3.2. Confluence of $\overline{\lambda}\mu\widetilde{\mu}_T$ calculus. Next, we define the term G^* which is obtained from G by simultaneously reducing all the existing redexes of the term G.

Definition 3.3. Let G be an arbitrary term of $\overline{\lambda}\mu\widetilde{\mu}_T$. The term G^* is defined inductively as follows:

$(*1_n)$ $x^* \equiv x$ $\quad(*2_n)$ $(\lambda x.v)^* \equiv \lambda x.v^*$ $\quad(*3_n)$ $(\mu\alpha.c)^* \equiv \mu\alpha.c^*$
$(*4_n)$ $\alpha^* \equiv \alpha$ $\quad(*5_n)$ $(v \bullet E)^* \equiv v^* \bullet E^*$ $\quad(*6_n)$ $(\widetilde{\mu}x.c)^* \equiv \widetilde{\mu}x.c^*$
$(*7_n)$ $(\langle v \parallel e\rangle)^* \equiv \langle v^* \parallel e^*\rangle$ if $\langle v \parallel e\rangle \neq \langle \lambda x.v_1 \parallel v_2 \bullet E\rangle$, $\langle v \parallel e\rangle \neq \langle \mu\alpha.c \parallel E\rangle$ and $\langle v \parallel e\rangle \neq \langle v \parallel \widetilde{\mu}x.c\rangle$
$(*8_n)$ $(\langle \lambda x.v_1 \parallel v_2 \bullet E\rangle)^* \equiv \langle v_1^*[x \leftarrow v_2^*] \parallel E^*\rangle$
$(*9_n)$ $(\langle \mu\alpha.c \parallel E\rangle)^* \equiv c^*[\alpha \leftarrow E^*]$
$(*10_n)$ $(\langle v \parallel \widetilde{\mu}x.c\rangle)^* \equiv c^*[x \leftarrow v^*]$

Theorem 3.1 (Star property for $\Rightarrow_n$). *If $G \Rightarrow_n G'$ then $G' \Rightarrow_n G^*$.*

Proof. Induction on the structure of G. We show only one illustrative case when $G = \langle \lambda x.v_1 \parallel v_2 \bullet E\rangle$.

If $\langle \lambda x.v_1 \parallel v_2 \bullet E\rangle \Rightarrow_n G'$ then we distinguish two subcases:

1. $G' = \langle \lambda x.v_1' \parallel v_2' \bullet E'\rangle$ for some v_1', v_2', and E' such that $v_i \Rightarrow_n v_i'$, $i = 1, 2$ and $E \Rightarrow_n E'$. By the induction hypothesis, $v_i' \Rightarrow_n v_i^*$, $i = 1, 2$ and $E' \Rightarrow_n E^*$. Then, $G' = \langle \lambda x.v_1' \parallel v_2' \bullet E'\rangle \Rightarrow_n \langle v_1^*[x \leftarrow v_2^*] \parallel E^*\rangle = G^*$ by $(g8_n)$.

2. $G' = \langle v_1'[x \leftarrow v_2'] \parallel E'\rangle$ for some v_1', v_2', and E' such that $v_i \Rightarrow_n v_i'$, $i = 1, 2$ and $E \Rightarrow_n E'$. By the induction hypothesis, $v_i' \Rightarrow_n v_i^*$, $i = 1, 2$ and $E' \Rightarrow_n E^*$. Then, $G' = \langle v_1'[x \leftarrow v_2'] \parallel E'\rangle \Rightarrow_n \langle v_1^*[x \leftarrow v_2^*] \parallel E^*\rangle$ by Lemma 3.3(3) and $(g7_n)$. □

Now it is easy to deduce the diamond property for $\Rightarrow_n$.

Theorem 3.2 (Diamond property for $\Rightarrow_n$).
If $G_1 \;{}_n\!\Leftarrow G \Rightarrow_n G_2$ then $G_1 \Rightarrow_n G' \;{}_n\!\Leftarrow G_2$ for some G'.

Finally, from Theorem 3.2, it follows that $\overline{\lambda}\mu\widetilde{\mu}_T$ is confluent.

Theorem 3.3 (Confluence of $\overline{\lambda}\mu\widetilde{\mu}_T$).
If $G_1 \;{}_n\!\twoheadleftarrow G \twoheadrightarrow_n G_2$ then $G_1 \twoheadrightarrow_n G' \;{}_n\!\twoheadleftarrow G_2$ for some G'.

4. Continuation semantics

4.1. Introduction. When interpreting the calculi that embody a notion of control, it is convenient to start from continuation semantics that enables to explicitly refer to *continuations*, the semantic constructs that represent evaluation contexts.

The method of continuations was first introduced in [**37**] in order to formalize a flow control in programming languages. Continuations can be seen as analogues of the evaluation contexts, used to evaluate terms. Hence, the term is evaluated in the context representing the rest of the computation. A subterm is evaluated in a new context where the rest of the term is evaluated, and then handed to the old context. The value obtained by evaluation of the term is passed to the context.

Continuation-passing-style (cps) translations were introduced by Fischer and Reynolds in [**9**] and [**31**] for the call-by-value lambda calculus, whereas a call-by-name variant was introduced by Plotkin in [**28**]. Moggi gave a semantic version of a call-by-value cps translation in his study of notions of computation in [**25**]. Lafont [**19**] introduced a cps translation of the call-by-name $\lambda\mathcal{C}$ calculus [**7**,**8**] to a fragment of lambda calculus that corresponds to the $\neg, \wedge$-fragment of the intuitionistic logic. Hence, continuation semantics can be seen as a generalization of the double negation rule from logic, in a sense that cps translation is a transformation on terms which, when observed on types, corresponds to a double negation translation.

In Selinger in [**34**], categorical semantics for both, call-by-name and call-by-value versions of Parigot's $\lambda\mu$ calculus [**27**] with disjunction types was given. In this work the notion of *control category* is formally introduced. It is shown that the call-by-name $\lambda\mu$ calculus forms an internal language for control categories, whereas the call-by-value $\lambda\mu$ calculus forms an internal language for co-control categories. The opposite of the call-by-name model is shown to be equivalent to the cbv model in the presence of product and disjunction types. Hofmann and Streicher presented categorical continuation models for the call-by-name $\lambda\mu$ calculus in [**17**] and showed the completeness.

In their original work on the $\overline{\lambda}\mu\widetilde{\mu}$ calculus [**6**], Curien and Herbelin defined a call-by-name and a call-by-value cps-translations of the complete *typed* $\overline{\lambda}\mu\widetilde{\mu}$ calculus into simply typed lambda calculus. The important point to notice is that they also interpret the types of the form $A - B$, which are dual to the arrow types $A \to B$. The translations validate call-by-name and call-by-value discipline, respectively.

Lengrand gave categorical semantics of the *typed* $\overline{\lambda}\mu\widetilde{\mu}$ calculus and the $\lambda\xi$ calculus (implicational fragment of the classical sequent calculus LK) in [**20**].

Ong [**26**] defined a class of categorical models for the call-by-name $\lambda\mu$ calculus based on fibrations. This model was later extended for two forms of disjunction by Pym and Ritter in [**30**].

4.2. Category of continuations. *Categories of continuations* were introduced by Hofmann and Streicher in [**16**]. They can be seen as special instances of *control categories*, which were introduced and formally described by Selinger in [**34**]. In simple words, control categories are cartesian closed categories enriched with premonoidal structure of [**29**].

Let $\mathcal{C}$ be a category with distributive finite products and sums. We also assume that there is a fixed object $R \in \mathcal{C}$ such that exponentials of the form R^A exist for all objects A. If $\mathcal{C}$ also satisfies the mono requirement (i.e., the morphism $\partial_A : A \to R^{R^A}$ is monic[2] for all $A \in \mathcal{C}$) then such a category $\mathcal{C}$ is called a *response category* and R is called an *object of responses*.

For a given response category $\mathcal{C}$, the full subcategory of $\mathcal{C}$ that consists of the objects of the form R^A is called a *category of continuations* and is denoted by $R^{\mathcal{C}}$. This category is cartesian closed [**22**] and has a canonical premonoidal structure [**34**]. This can be summarized as follows:

[2]A morphism $f : A \to B$ in a category $\mathcal{C}$ is called *monic* if for any object C and any two morphisms $g_1, g_2 : C \to A$, if $f \circ g_1 = f \circ g_2$, then $g_1 = g_2$.

- $1 \cong R^0$ (terminal object in $R^{\mathcal{C}}$ is 1)
- $R^A \times R^B \cong R^{A+B}$ ($R^{\mathcal{C}}$ has cartesian products)
- $(R^B)^{(R^A)} \cong R^{R^A \times B}$ ($R^{\mathcal{C}}$ has exponentials)
- $\perp := R^1 \cong R$ (bottom exists in $R^{\mathcal{C}}$)
- $R^A ⅋ R^B := R^{A \times B}$ ($⅋$ is a binoidal functor in $R^{\mathcal{C}}$, see [**34**] for details).

In fact, it is proved in [**34**] that every control category is equivalent to a category of continuations (see also [**11**]).

4.3. Category of negated domains. As a further specialization of categories of continuations, we describe the category of negated domains $\mathcal{N}_R$ which was introduced by Lafont in [**19**], where he investigated the translation of classical propositional logic to the $\neg, \wedge$-fragment of the intuitionistic propositional logic.

Before giving the formal definition, let us first of all, fix some basic terminology that will be used.

- A *predomain* is a partial order where all directed subsets have a supremum. It does not necessarily have a least element.
- A *domain* is a predomain with a least element called bottom, denoted by $\perp$.
- A *Scott continuous function* is a monotone function that preserves suprema of directed sets.
- A *strict continuous function* is a function that preserves bottom elements.

The category of predomains and Scott continuous functions is denoted by $\mathcal{P}$. The category of domains and Scott continuous (resp. strict Scott continuous) functions is denoted by $\mathcal{D}$ (resp. $\mathcal{D}_\perp$).

Let R be some fixed domain with the bottom $\perp_{\mathrm{R}}$ for a category $\mathcal{D}$. We call R a *domain of responses.* For each predomain $A \in \mathcal{P}$ we can form the exponential $R^A \in \mathcal{D}$. Then we define the category $\mathcal{N}_R$ equivalent to a full subcategory of $\mathcal{D}$, where the morphisms operate on exponentials of the form R^A. Hence, the category $\mathcal{N}_R$ is an instance of the category of continuations where the category $\mathcal{P}$ of predomains is a basic category, since it has finite products and sums, and exponentials of the form R^A exist. Let us give the formal definition.

Definition 4.1. The *category of negated domains* $\mathcal{N}_R$ over the category $\mathcal{P}$ of predomains is defined as follows:

- the objects of $\mathcal{N}_R$ are objects of $\mathcal{P}$ (predomains),
- $\mathcal{N}_R(A, B) = \mathcal{P}(R^A, R^B)$,
- composition of morphisms in $\mathcal{N}_R$ is inherited from $\mathcal{P}$.

As already mentioned, $(R^B)^{(R^A)} \cong R^{R^A \times B}$, so we will denote the function space operator in $\mathcal{N}_R$ as (see [**36**]): $A \Rightarrow B := R^A \times B$.

Since by the assumption, R has a bottom element, all the exponentials have bottom elements. The bottom element for R^A is given by $\perp_{R^A} = \lambda x : A.\perp_{\mathrm{R}}$ for any predomain $A \in \mathcal{P}$. Hence, $\mathcal{N}_R$ is equivalent to a full subcategory of $\mathcal{D}$. The least fixpoint for $f \in \mathcal{N}_R(A, A)$ is given by $\bigsqcup_{n \in \mathbb{N}} f^n(\perp_{R^A})$. The following theorem (proved in [**35**]) states that the category $\mathcal{N}_R$ has enough structure to interpret functional calculi, especially the calculi with control operators.

THEOREM 4.1. *The category $\mathcal{N}_R$ is cartesian closed and has the least fixpoint operator, for any domain R.*

4.4. From ordinary models to continuation models. For the extensional lambda calculus, a model is given by an object C in a cartesian closed category, such that C is isomorphic to its function space, i.e., $C \cong C^C$ [**32**, **33**]. We call such an object a *reflexive object*. (For solutions of recursive domain equations, we refer the reader to [**1**, **10**, **14**].)

In order to obtain a model of lambda calculus and its extensions in $\mathcal{N}_R$, we have the same requirement in the category $\mathcal{N}_R$ of negated domains, which means that we are looking for an object K such that $K \cong K \Rightarrow K$. This requires solving the domain equation $K \cong R^K \times K$ in $\mathcal{P}$. For K which is the initial solution of this domain equation, we have that $R^K \cong R^{R^K \times K} \cong (R^K)^{(R^K)}$, so we conclude that $C = R^K$ is a solution of domain equation $C \cong C^C$ in $\mathcal{P}$ and is called a *continuation model* of the untyped lambda calculus. In [**35**] it is proved that $C = R^K$ is isomorphic to Scott's C_∞ model of extensional lambda calculus [**32**, **33**] by taking $C = R$.

We can interpret the untyped lambda calculus in $\mathcal{N}_R$. The meaning of a lambda term is a morphism R^K mapping continuations (the elements of K) to responses (the elements of R). The continuation for the function $f: R^A \to R^B$ is a pair $\langle s, k\rangle$, where the argument for the function f is $s \in R^A$ and $k \in B$ is the continuation for $f(s)$.

This interpretation can be extended to Felleisen's call-by-name $\lambda\mathcal{C}$ calculus [**7**] and to the untyped version of Parigot's $\lambda\mu$ calculus [**27**], given in Streicher and Reus [**35**]. In the same work it is also proved that the semantic equations defining the interpretations in the continuation model of the untyped lambda calculus are in 1-1 correspondence with the transition rules of Krivine's abstract machine.

4.5. Kleisli category. Kleisli categories introduced by Kleisli in [**18**] (see also [**5**, **23**]) provide the categorical semantics of computations based on monads. Since every monad corresponds to Kleisli triple, the semantics can be given using Kleisli triples that are easier to justify computationally.

When interpreting a programming language in the call-by-value setting in a category $\mathcal{C}$, we need to distinguish the objects A that represent the values of type A from the objects TA that represent the computations of type A. The computations of type A are obtained by applying a functor T to A, which is called the *notion of computation* [**25**]. There are certain conditions that T has to satisfy and it turns out that T needs to give rise to a Kleisli triple, whereas programs form the Kleisli category for such a triple.

The following definitions are taken from Moggi's paper on notions of computations [**25**], which are in turn taken from [**24**].

DEFINITION 4.2. A *Kleisli triple* over a category $\mathcal{C}$ is a triple $(T, \eta, \_^*)$, such that $T : \mathrm{Obj}(\mathcal{C}) \to \mathrm{Obj}(\mathcal{C})$, $\eta_A : A \to TA$ for $A \in \mathrm{Obj}(\mathcal{C})$, $f^* : TA \to TB$ for

$f : A \to TB$ and the following equations hold:

$$\eta_A^* = id_{TA}; \qquad f^* \circ \eta_A = f \text{ for } f : A \to TB;$$
$$g^* \circ f^* = (g^* \circ f)^* \text{ for } f : A \to TB \text{ and } g : B \to TK.$$

Next we give the definition of the *Kleisli category*.

DEFINITION 4.3. The *Kleisli category* $\mathcal{C}_T$ over a category $\mathcal{C}$ for a given Kleisli triple $(T, \eta, \_^*)$ is defined as follows:

- the objects of $\mathcal{C}_T$ are the objects of $\mathcal{C}$;
- $\mathcal{C}_T(A, B) = \mathcal{C}(A, TB)$;
- $\mathrm{id}_{\mathcal{C}_T} = \eta_A : A \to TA$;
- $g \circ_{\mathcal{C}_T} f = g^* \circ f : A \to TD$ for $f \in \mathcal{C}_T(A, B)$ and $g \in \mathcal{C}_T(B, D)$.

4.6. Kleisli triple of continuations. Depending on the specific computation that we want to model, different computational monads or Kleisli triples can be chosen. In this analysis we will consider *Kleisli triple of continuations* given by

- $TA = R^{R^A}$, where R is the fixed object of responses, together with
- a family of morphisms $\eta_A(a) = \lambda k : R^A.k(a)$ and
- an operation on morphisms
  $f^*(s) = \lambda k : R^B.s(\lambda a : A.f(a)(k))$ for $f : A \to TB$ and $s \in TA$.

We will denote by $\mathcal{K}_R$ the Kleisli category over the category $\mathcal{P}$ of predomains for a given Kleisli triple of continuations $(T, \eta, \_^*)$.

Then, the intuitive meaning of η_A is the inclusion of values into computations, whereas f^* can be seen as an extension of a function f mapping values to computations into a function mapping computations into computations.

As noticed in [**36**], the Kleisli category $\mathcal{K}_R$ for a continuation Kleisli triple and the dual of the category of negated domains $\mathcal{N}_R^{op}$ are isomorphic and the isomorphism is given by $H : \mathcal{K}_R \to \mathcal{N}_R^{op}$ and $K : \mathcal{N}_R^{op} \to \mathcal{K}_R$, where

$$H(f) = \lambda k : R^B.\lambda v : A.f(v)(k) \in (R^A)^{(R^B)} \text{ for } f \in (TB)^A,$$
$$K(g) = \lambda v : A.\lambda k : R^B.g(k)(v) \in (TB)^{(A)} \quad \text{for } g \in (R^A)^{R^B}.$$

5. Semantics

As we have seen, the categories $\mathcal{N}_R$ and $\mathcal{C}_T$ are very convenient for defining the semantics of the various calculi with control operators, since they allow to explicitly deal with continuations. Therefore, we think they are a good starting point for giving the semantics of $\overline{\lambda}\mu\widetilde{\mu}$ calculus, although the results of this section apply to any categorical model in which it is possible to solve the domain equations.

As mentioned previously, $\overline{\lambda}\mu\widetilde{\mu}$ is not confluent due to the presence of the critical pair $\langle \mu\alpha.c \parallel \widetilde{\mu}x.c \rangle$. Hence, we consider separately two well-behaved subsyntaxes closed either under call-by-value ($\overline{\lambda}\mu\widetilde{\mu}_Q$) or under call-by-name reduction ($\overline{\lambda}\mu\widetilde{\mu}_T$).

5.1. Semantics of $\overline{\lambda}\mu\widetilde{\mu}_Q$ calculus. In this subection we will consider $\overline{\lambda}\mu\widetilde{\mu}_Q$, which is a variant of untyped $\overline{\lambda}\mu\widetilde{\mu}$ calculus closed under the call-by-value reduction.

We give the definition of the interpretation functions for all four syntactic categories of the calculus. Having an interpretation function also for the values prevents the values and the computations to be confused. Lambda abstractions are values, but can also have arguments that are values, producing computations as the result, so it is necessary to have $W \cong C^W$.

Definition 5.1. Let us consider the initial solution of the system of domain equations $W \cong C^W$, $K \cong R^W$, $C \cong R^K$. Let Env be the set of the environments that map the caller variables to the elements of W and the callee variables to the elements of K, i.e., for $\rho \in \mathsf{Env}$: $\forall x \in \mathrm{Var}_v, \rho(x) \in W$ and $\forall \alpha \in \mathrm{Var}_e, \rho(\alpha) \in K$. The interpretation functions

$$\llbracket - \rrbracket_{\mathrm{V}} : \mathsf{Value} \to \mathsf{Env} \to W = C^W \qquad \llbracket - \rrbracket_{\mathrm{D}} : \mathsf{Caller} \to \mathsf{Env} \to C = R^K$$
$$\llbracket - \rrbracket_{\mathrm{C}} : \mathsf{Callee} \to \mathsf{Env} \to K = R^W \qquad \llbracket - \rrbracket_{\mathrm{R}} : \mathsf{Capsule} \to \mathsf{Env} \to R$$

are defined as follows:

Value:

$$\llbracket x \rrbracket_{\mathrm{V}}\rho = \rho(x)$$
$$\llbracket \lambda x.v \rrbracket_{\mathrm{V}}\rho = \lambda w.\llbracket v \rrbracket_{\mathrm{D}}\rho[x := w]$$

Callee:

$$\llbracket \alpha \rrbracket_{\mathrm{C}}\rho = \rho(\alpha)$$
$$\llbracket V \bullet e \rrbracket_{\mathrm{C}}\rho = \lambda w.(w(\llbracket V \rrbracket_{\mathrm{V}}\rho))(\llbracket e \rrbracket_{\mathrm{C}}\rho)$$
$$\llbracket \widetilde{\mu}x.c \rrbracket_{\mathrm{C}}\rho = \lambda w.\llbracket c \rrbracket_{\mathrm{R}}\rho[x := w]$$

Caller:

$$\llbracket x \rrbracket_{\mathrm{D}}\rho = \lambda k.k\llbracket x \rrbracket_{\mathrm{V}}\rho$$
$$\llbracket \lambda x.v \rrbracket_{\mathrm{D}}\rho = \lambda k.k\llbracket \lambda x.v \rrbracket_{\mathrm{V}}\rho$$
$$\llbracket \mu\alpha.c \rrbracket_{\mathrm{D}}\rho = \lambda k.\llbracket c \rrbracket_{\mathrm{R}}\rho[\alpha := k]$$

Capsule:

$$\llbracket \langle v \parallel e \rangle \rrbracket_{\mathrm{R}}\rho = \llbracket v \rrbracket_{\mathrm{D}}\rho(\llbracket e \rrbracket_{\mathrm{C}}\rho)$$

We will omit the subscripts in various interpretations, (since they can be deduced from the term being interpreted), a part from $\llbracket - \rrbracket_{\mathrm{V}}$ where we leave the subscript to avoid the ambiguity.

One important difference when interpreting the call-by-value calculus (with respect to the interpretation of the call-by-name variant) is that variables are interpreted as values, i.e., $\rho(x) \in W$, whereas in the call-by-name case variables are interpreted as computations, i.e., $\rho(x) \in C$.

The different syntactic constructs of $\overline{\lambda}\mu\widetilde{\mu}_Q$ can be seen as elements of the following semantical objects:

- values are the elements of W,
- callers as computations are the elements of $C = R^{R^W}$,
- callees as continuations are the elements of $K = R^W$,
- capsules as responses are the elements of R.

In the case of callees, $V \bullet e$ and $\widetilde{\mu}x.c$ can be seen as call-by-value evaluation contexts. For $V \bullet e$, the computation (seen as a value) is applied to V and then evaluated in the evaluation context e. For $\widetilde{\mu}x.c$, the caller is just fed into the capsule c. In the case of capsules, the meaning of the term v (element of C) is applied to the continuation bound to e (element of K) and produces an element of R.

Also, notice that the interpretation of values in C is obtained by applying $\eta_A(a) = \lambda k : R^A.k(a)$ from the Kleisli triple, to the interpretation of values in W. Hence, we include values into denotations. On the other hand, $\mu\alpha.c$ is not a value, so its interpretation is given only in C. Its meaning is the functional abstraction over the continuation variable α.

Next we give some lemmas that are used to prove that the semantics is preserved under the reduction rules.

LEMMA 5.1 (Substitution lemma). *Let G be the term of $\overline{\lambda}\mu\widetilde{\mu}_Q$ (caller, callee, or capsule). Then*

1. $[\![G[x \leftarrow V]]\!]\rho = [\![G]\!]\rho[x := [\![V]\!]_V\rho]$, *for all four interpretation functions.*
2. $[\![G[\alpha \leftarrow e]]\!]\rho = [\![G]\!]\rho[\alpha := [\![e]\!]\rho]$.

PROOF. 1. Induction on the structure of V, followed by induction on the structure of G.

1.1. $[\![G[x \leftarrow y]]\!]\rho = [\![G]\!]\rho[x := \rho(y)]$ We prove the statement only for $[\![-]\!]_{\mathrm{D}}$.

* $G = z$ trivial
* $G = x$ $[\![x[x \leftarrow y]]\!]\rho = [\![y]\!]\rho = \lambda k.k\rho(y) = \lambda k.k\rho[x := \rho(y)](x)$ $= [\![x]\!]\rho[x := \rho(y)]$
* $G = \lambda z.r$ $[\![\lambda z.r[x \leftarrow y]]\!]\rho = \lambda k.k(\lambda w.[\![r[x \leftarrow y]]\!]\rho[z := w])$ $= \lambda k.k(\lambda w.[\![r]\!]\rho[x := \rho(y), z := w]) = [\![\lambda z.r]\!]\rho[x := \rho(y)]$
* $G = \mu\alpha.c$ $[\![\mu\alpha.c[x \leftarrow y]]\!]\rho = \lambda k.[\![c[x \leftarrow y]]\!]\rho[\alpha := k]$ $= \lambda k.[\![c]\!]\rho[x := \rho(y), \alpha := k] = [\![\mu\alpha.c]\!]\rho[x := \rho(y)]$
* The cases $G = \beta$, $G = V \bullet e$, $G = \widetilde{\mu}y.c$, and $G = \langle v \parallel e\rangle$ are either trivial or follow from the induction hypothesis.

1.2. $[\![G[x \leftarrow \lambda y.v]]\!]\rho = [\![G]\!]\rho[x := [\![\lambda y.v]\!]_V\rho]$ Again, we prove the statement for $[\![-]\!]_{\mathrm{D}}$.

* $G = z$ trivial
* $G = x$ $[\![x[x \leftarrow \lambda y.v]]\!]\rho = [\![\lambda y.v]\!]\rho = \lambda k.k([\![\lambda y.v]\!]_V\rho)$ $= \lambda k.k(\rho[x := [\![\lambda y.v]\!]_V\rho](x)) = [\![x]\!]\rho[x := [\![\lambda y.v]\!]_V\rho]$
* $G = \lambda z.r$ $[\![\lambda z.r[x \leftarrow \lambda y.v]]\!]\rho = \lambda k.k([\![\lambda z.r[x \leftarrow \lambda y.v]]\!]_V\rho)$ $= \lambda k.k(\lambda w.[\![r[x \leftarrow \lambda y.v]]\!]\rho[z := w])$ $= \lambda k.k(\lambda w.[\![r]\!]\rho[x := [\![\lambda y.v]\!]_V\rho, z := w]) = [\![\lambda z.r]\!]\rho[x := [\![\lambda y.v]\!]_V\rho]$
* $G = \mu\alpha.c$ $[\![\mu\alpha.c[x \leftarrow \lambda y.v]]\!]\rho = \lambda k.[\![c[x \leftarrow \lambda y.v]]\!]\rho[\alpha := k]$ $= \lambda k.[\![c]\!]\rho[x := [\![\lambda y.v]\!]_V\rho][\alpha := k] = [\![\mu\alpha.c]\!]\rho[x := [\![\lambda y.v]\!]_V\rho]$
* The cases $G = \beta$, $G = V \bullet e$, $G = \widetilde{\mu}y.c$, and $G = \langle v \parallel e\rangle$ again follow trivially.

2. Induction on the structure of G, followed by induction on the structure of e. It is enough to prove the lemma for $G = \alpha$ because all the other cases follow either trivially ($G = \gamma$, $G = x$) or by the induction hypothesis.

* $e = \beta$ $[\![\alpha[\alpha \leftarrow \beta]]\!]\rho = [\![\beta]\!]\rho = [\![\alpha]\!]\rho[\alpha := Ai\beta\rho]$
* $e = \widetilde{\mu}x.c$ $[\![\alpha[\alpha \leftarrow \widetilde{\mu}x.c]]\!]\rho = [\![\widetilde{\mu}x.c]\!]\rho = [\![\alpha]\!]\rho[\alpha := [\![\widetilde{\mu}x.c]\!]\rho]$
* $e = V \bullet e$ $[\![\alpha[\alpha \leftarrow V \bullet e]]\!]\rho = [\![V \bullet e]\!]\rho = [\![\alpha]\!]\rho[\alpha := [\![V \bullet e]\!]\rho]$. □

THEOREM 5.1 (Preservation of the semantics under reduction).

If $G_1 \to G_2$, then $[\![G_1]\!] = [\![G_2]\!]$.

PROOF. 1. $\langle \mu\alpha.c \parallel e\rangle \to c[\alpha \leftarrow e]$

$$[\![\langle \mu\alpha.c \parallel e\rangle]\!]\rho = [\![\mu\alpha.c]\!]\rho([\![e]\!]\rho) = (\lambda k.[\![c]\!]\rho[\alpha := k])([\![e]\!]\rho)$$
$$= [\![c]\!]\rho[\alpha := [\![e]\!]\rho] = [\![c[\alpha \leftarrow e]]\!]\rho$$

2. $\langle V \parallel \widetilde{\mu}x.c\rangle \to c[x \leftarrow V]$ Induction on the structure of V.

* $V = y$ $[\![\langle y \parallel \widetilde{\mu}x.c\rangle]\!]\rho = [\![y]\!]\rho([\![\widetilde{\mu}x.c]\!]\rho) = (\lambda k.k\rho(y))(\lambda w.[\![c]\!]\rho[x := w])$
$$= (\lambda w.[\![c]\!]\rho[x := w])\rho(y) = [\![c]\!]\rho[x := \rho(y)] = [\![c[x \leftarrow y]]\!]\rho$$

* $V = \lambda y.v$ $[\![\langle \lambda y.v \parallel \widetilde{\mu}x.c\rangle]\!]\rho = [\![\lambda y.v]\!]\rho([\![\widetilde{\mu}x.c]\!]\rho)$
$$= (\lambda k.k(\lambda w.[\![v]\!]\rho[y := w]))(\lambda w_1.[\![c]\!]\rho[x := w_1])$$
$$= (\lambda w_1.[\![c]\!]\rho[x := w_1])(\lambda w.[\![v]\!]\rho[y := w])$$
$$= [\![c]\!]\rho[x := \lambda w.[\![v]\!]\rho[y := w]] = [\![c[x \leftarrow \lambda y.v]]\!]\rho$$

Hence $[\![\langle V \parallel \widetilde{\mu}x.c\rangle]\!]\rho = [\![c[x \leftarrow V]]\!]\rho$.

3. $\langle \lambda x.v \parallel V \bullet e\rangle \to \langle V \parallel \widetilde{\mu}x.\langle v \parallel e\rangle\rangle$

$$[\![\langle \lambda x.v \parallel V \bullet e\rangle]\!]\rho = [\![\lambda x.v]\!]\rho([\![V \bullet e]\!]\rho)$$
$$= (\lambda k.k(\lambda w.[\![v]\!]\rho[x := w]))(\lambda w_1.(w_1([\![V]\!]_{\mathrm{V}}\rho))([\![e]\!]\rho))$$
$$= (\lambda w_1.(w_1([\![V]\!]_{\mathrm{V}}\rho))([\![e]\!]\rho))(\lambda w.[\![v]\!]\rho[x := w])$$
$$= (\lambda w.[\![v]\!]\rho[x := w])([\![V]\!]_{\mathrm{V}}\rho)([\![e]\!]\rho)$$
$$= [\![v]\!]\rho[x := [\![V]\!]_{\mathrm{V}}\rho]([\![e]\!]\rho) = [\![v[x \leftarrow V]]\!]\rho([\![e]\!]\rho)$$
$$[\![\langle V \parallel \widetilde{\mu}x.\langle v \parallel e\rangle\rangle]\!]\rho = [\![V]\!]\rho([\![\widetilde{\mu}x.\langle v \parallel e\rangle]\!]\rho)$$
$$= [\![\langle v \parallel e\rangle[x \leftarrow V]]\!]\rho \quad \text{as in 2.}$$
$$= [\![v[x \leftarrow V]]\!]\rho([\![e]\!]\rho) \quad \text{since } x \notin e.$$
□

For the complete proofs see [**21**].

5.2. Semantics of $\overline{\lambda}\mu\widetilde{\mu}_T$ calculus. In Subection 5.1, we considered two different types of computations, namely the *values* as the elements of W and the *computations* as the elements of C. With the help of $\eta_A(a) = \lambda k : R^A.k(a)$ from the Kleisli triple, we had a way of including the values into the computations. So we will apply the same technique at the level of *continuations*. In the set of callees we will distinguish basic continuations that we call *co-values* (called *applicative contexts* in [**6**]), from the rest of continuations.

Next, we give the interpretation functions for all the *four* syntactic constructs of $\overline{\lambda}\mu\widetilde{\mu}_T$. Giving the interpretation function also for co-values, makes a clear difference between co-values and the rest of callees.

DEFINITION 5.2. Let K be the initial solution of the domain equation $K \cong R^K \times K$ and let $C = R^K$ and $F = R^C$. With Env we denote the set of the environments that map the caller variables to the elements of C and the callee variables to the elements of K, i.e., for $\rho \in \mathsf{Env}$:

$$\forall x \in \mathrm{Var}_v, \rho(x) \in C \quad \text{and} \quad \forall \alpha \in \mathrm{Var}_e, \rho(\alpha) \in K.$$

Then the interpretation functions are defined as follows:

$[\![-]\!]_{\mathrm{C}}: \mathsf{Co\text{-}value} \to \mathsf{Env} \to K \qquad [\![-]\!]_{\mathrm{E}}: \mathsf{Callee} \to \mathsf{Env} \to F = R^C$

$[\![-]\!]_{\mathrm{D}}: \mathsf{Caller} \to \mathsf{Env} \to C = R^K \qquad [\![-]\!]_{\mathrm{R}}: \mathsf{Capsule} \to \mathsf{Env} \to R$

Co-value :

$$[\![\alpha]\!]_{\mathrm{C}}\rho = \rho(\alpha)$$
$$[\![v \bullet E]\!]_{\mathrm{C}}\rho = \langle [\![v]\!]_{\mathrm{D}}\rho, [\![E]\!]_{\mathrm{C}}\rho\rangle$$

Caller :

$$[\![x]\!]_{\mathrm{D}}\rho = \rho(x)$$
$$[\![\lambda x.v]\!]_{\mathrm{D}}\rho = \lambda\langle s,k\rangle.[\![v]\!]_{\mathrm{D}}\rho[x := s]k$$
$$[\![\mu\alpha.c]\!]_{\mathrm{D}}\rho = \lambda k.[\![c]\!]_{\mathrm{R}}\rho[\alpha := k]$$

Callee :

$$[\![\alpha]\!]_{\mathrm{E}}\rho = \lambda s.s([\![\alpha]\!]_{\mathrm{C}}\rho)$$
$$[\![v \bullet E]\!]_{\mathrm{E}}\rho = \lambda s.s([\![v \bullet E]\!]_{\mathrm{C}}\rho)$$
$$[\![\widetilde{\mu}x.c]\!]_{\mathrm{E}}\rho = \lambda s.[\![c]\!]_{\mathrm{R}}\rho[x := s]$$

Capsule :

$$[\![\langle v \parallel e\rangle]\!]_{\mathrm{R}}\rho = [\![e]\!]_{\mathrm{E}}\rho([\![v]\!]_{\mathrm{D}}\rho)$$

We will omit the subscripts in various interpretations, as they can be deduced from the terms being interpreted, apart from $[\![-]\!]_{\mathrm{E}}$ where we leave the subscript to avoid the ambiguity.

Now, we can see the different syntactic constructs of $\overline{\lambda}\mu\widetilde{\mu}_T$ as the elements of the following semantical objects:

- callers as computations are the elements of $C = R^K$,
- co-values as basic continuations are the elements of $K \cong R^K \times K$,
- callees as continuations are the elements of $F = R^C$,
- capsules as responses are the elements of R.

Since $K \cong R^K \times K$, continuations are of the form $\langle s,k\rangle$, where $s \in C$ and $k \in K$. Therefore we can see continuations as infinite lists of denotations which correspond to the denotational versions of the call-by-name evaluation contexts. Callers are interpreted as functions that map continuations to responses. This reflects the fact that a caller can either get the data from a callee or ask it to replace one of its internal callee variables. Hence, callers expect callees as arguments. Since a callee can ask a caller to replace one of its internal caller variables, it has to have a functional part that could be applied to a caller. Finally, in the case of capsules, the interpretation of the callee is applied to the interpretation of the caller, thus producing an element in R.

Also, notice that the interpretation of the co-values in F is obtained by applying $\eta_K(k) = \lambda s : R^{R^K}.s(k)$ from the Kleisli triple, to the interpretation of the co-values in K. Hence, we include the co-values into the continuations. On the other hand, $\widetilde{\mu}x.c$ is not a co-value, hence its interpretation is given only in F.

As in the previous section, we first give some lemmas that are used later to prove that the semantics is preserved under the reduction rules.

Lemma 5.2 (Substitution lemma). *Let G be the term of $\overline{\lambda}\mu\widetilde{\mu}_T$ (caller, callee, or capsule). Then*

1. $[\![G[x \leftarrow v]]\!]\rho = [\![G]\!]\rho[x := [\![v]\!]\rho]$;
2. $[\![G[\alpha \leftarrow E]]\!]_{(K)}\rho = [\![G]\!]_{(K)}\rho[x := [\![E]\!]_C\rho]$

where $[\![-]\!]_{(K)}$ *means that in the case of co-values, the lemma holds for both interpretations, namely* $[\![-]\!]_E$ *and* $[\![-]\!]_C$.

THEOREM 5.2 (Preservation of the semantics under reduction).

$$\text{If } G_1 \to G_2 \text{ then } [\![G_1]\!] = [\![G_2]\!]$$

6. Conclusions

This work investigates some properties of $\overline{\lambda}\mu\widetilde{\mu}_T$ and $\overline{\lambda}\mu\widetilde{\mu}_Q$, the two subcalculi of untyped $\overline{\lambda}\mu\widetilde{\mu}$ calculus of Curien and Herbelin [**6**], closed under the call-by-name and the call-by-value reduction, respectively.

First of all, the proof of confluence for both versions of the $\overline{\lambda}\mu\widetilde{\mu}$ calculus is given, adopting the method of parallel reduction given in [**38**]. As a step towards a better understanding of denotational semantics of $\overline{\lambda}\mu\widetilde{\mu}$ calculus, its *untyped* call-by-value ($\overline{\lambda}\mu\widetilde{\mu}_Q$) and call-by-name ($\overline{\lambda}\mu\widetilde{\mu}_T$) versions are interpreted. Continuation semantics of $\overline{\lambda}\mu\widetilde{\mu}_Q$ and $\overline{\lambda}\mu\widetilde{\mu}_T$ is given using the category of negated domains of [**35**], and Moggi's Kleisli category over predomains for the continuation monad [**25**]. In both cases the reduction preserves the denotations.

The first future step to take is to explore completeness and show that these semantics are computationally adequate. We would also like to extend the present work to the complete symmetric calculus of [**6**] and find the interpretation for all the constructs of that calculus, including $e \bullet v$ and $\beta\lambda.e$. Still in the realm of categorical semantics, we intend to interprete the typed $\overline{\lambda}\mu\widetilde{\mu}$ calculus using fibrations, as done for the $\lambda\mu$ calculus in [**26**] and [**30**].

Acknowledgements. The authors would like to thank the two anonymous referees for their valuable remarks which helped us improve the final version of the paper.

References

1. R. M. Amadio and P-L. Curien, *Domains and Lambda-Calculi*, Cambridge University Press, Cambridge, 1998.
2. Z. M. Ariola and H. Herbelin, *Minimal classical logic and control operators*, in: *Proc. Annual International Colloquium on Automata, Languages and Programming ICALP '03*, Lect. Notes Comput. Sci. 2719, Springer-Verlag, Berlin, 2003, 871–885.
3. Z. M. Ariola, H. Herbelin, and A. Sabry, *A type-theoretic foundation of continuations and prompts*, in: *Proc. 9th Internat. Conf. on Funct. Prog. ICFP '04*, 40–53, 2004.
4. H. P. Barendregt, *The Lambda Calculus: its Syntax and Semantics*, North-Holland, Amsterdam, revised edition, 1984.
5. M. Barr and C. Wells, *Category Theory for Computing Science*, Les publications Centre de recherches mathématiques, 1999.
6. P.-L. Curien and H. Herbelin, *The Duality of Computation*, in: *Proc. 5th Internat. Conf. on Functional Programming, ICFP'00*, 233–243, Montreal, Canada, 2000; ACM Press.
7. M. Felleisen, D. P. Friedman, E. Kohlbecker and B. F. Duba, *A syntactic theory of sequential control*, Theor. Comput. Sci. 52(3):205–237, 1987.
8. M. Felleisen and R. Hieb, *The revised report on the syntactic theories of sequential control and state*, Theor. Comput. Sci. 103(2):235–271, 1992.
9. M. Fischer, *Lambda calculus schemata*, in: *Proc. ACM Conf. on Proving Assertions About Programs '72*, 104–109 ACM Press, 1972.

10. P. Freyd, *Remarks on algebraically compact categories*, in: *Notes of the London Mathematical Society*, 177, 1992.
11. C. Führmann, *The structure of call-by-value*, PhD thesis, University of Edinburgh, 2000.
12. S. Ghilezan and P. Lescanne, *Classical proofs, typed processes and intersection types*, in: *Proc. Workshop Types for Proofs and Programs TYPES '03 (Selected Papers)*, Lect. Notes Comput. Sci. 3085, 226–241, Springer-Verlag, Berlin, 2004.
13. T. Griffin, *A formulae-as-types notion of control*, in: *Proc. 19th Annual ACM Symp. on Principles Of Programming Languages, POPL '90*, 47–58, ACM Press, 1990.
14. C. A. Gunter and D. S. Scott, *Semantic domains*, In J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science*, volume B. Elsevier, Amsterdam, 1990.
15. W. Howard, *The formulae-as-types notion of construction*, in: J. P. Seldin and J. R. Hindley (eds.), *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, 479–490, Academic Press, 1980.
16. M. Hofmann and Th. Streicher, *Continuation models are universal for lambda-mu-calculus*, in: *Proc. 11th IEEE Annual Symposium on Logic in Computer Science LICS '97*, 387–397. IEEE Computer Society Press, 1997.
17. M. Hofmann and Th. Streicher, *Completeness of continuation models for $\lambda\mu$-calculus*, Inf. Comput. 179(2):332–355, 2002.
18. H. Kleisli, *Every standard construction is induced by a pair of adjoint functors*, in: *Proc. American Mathematical Society*, volume 16, 544–546, 1965.
19. Y. Lafont, *Negation versus implication*, Draft, 1991.
20. S. Lengrand, *Call-by-value, call-by-name, and strong normalization for the classical sequent calculus*, Electr. Notes Theor. Comput. Sci. 86, Elsevier, 2003.
21. S. Likavec, *Types for object-oriented and functional programming languages*, PhD thesis, Università di Torino, Italy, ENS Lyon, France, 2005.
22. Y. Lafont, B. Reus, and Th. Streicher, *Continuation semantics or expressing implication by negation*, Technical Report 93-21. University of Munich, 1993.
23. J. Lambek and P. J. Scott, *Introduction to higher order categorical logic*, Cambridge University Press, Cambridge, 1986.
24. E. Manes, *Algebraic theories*, in: *Graduate Texts in Mathematics*. Springer Verlag, Berlin, 1976.
25. E. Moggi, *Notions of computations and monads*, Inf. Comput. 93(1), 1991.
26. C.-H. L. Ong, *A semantic view of classical proofs: type-theoretic, categorical, denotational characterizations*, in: *Proc. 11th IEEE Annual Symposium on Logic in Computer Science LICS '97*, 230–241. IEEE Computer Society Press, 1997.
27. M. Parigot, *$\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction*, in: *Proc. Internat. Conf. on Logic Programming and Automated Reasoning, LPAR '92*, Lect. Notes Comput. Sci. 624, 190–201, Springer-Verlag, Berlin, 1992.
28. G. D. Plotkin, *Call-by-name, call-by-value and the λ-calculus*, Theor. Comput. Sci. 1:125–159, 1975.
29. J. Power and E. Robinson, *Premonoidal categories and notions of computation*, *Mathematical Structures in Computer Science*, 7(87):453–468, 1997.
30. D. Pym and E. Ritter, *On the semantics of classical disjunction*, J. Pure Appl. Algebra 159:315–338, 2001.
31. J. C. Reynolds, *Definitional interpreters for higher-order programming languages*, in: *Proc. ACM Annual Conference*, 717–740, ACM Press, 1972.
32. D. S. Scott, *Continuous lattices*, in: F. W. Lawvere (ed.), *Toposes, Algebraic Geometry and Logic*, Lect. Notes Math. 274, 97–136, Springer-Verlag, Berlin, 1972.
33. D. S. Scott, *Domains for denotational semantics*, in: M. Nielsen and E. M. Schmidt (ed), *Automata, Languages and Programming*, Lect. Notes Comput. Sci. 140, Springer-Verlag, Berlin, 1982, 577–613.
34. P. Selinger, *Control categories and duality: on the categorical semantics of the lambda-mu calculus*, Math. Struct. Comput. Sci. 11(2):207–260, 2001.

35. Th. Streicher and B. Reus, *Classical logic, continuation semantics and abstract machines*, J. Funct. Program. 8(6):543–572, 1998.
36. Th. Streicher and B. Reus, *Continuation semantics: Abstract machines and control operators*, Unpublished manuscript, 1998.
37. C. Strachey and C. P. Wadsworth, *Continuations: A mathematical semantics for handling full jumps*, Oxford University Computing Laboratory Technical Monograph PRG-11, 1974.
38. M. Takahashi, *Parallel reduction in λ-calculus*, Inf. Comput. 118:120–127, 1995.
39. P. Wadler, *Call-by-value is dual to call-by-name*, in: *Proc. 8th Internat. Conf. on Funct. Prog. ICFP '03*, 189–201, 2003.
40. P. Wadler, *Call-by-value is dual to call-by-name-reloaded*, in: *Proc. Conf. on Rewriting Technics and Applications RTA '05*, Lect. Notes Comput. Sci. 3467, 185–203, 2005.

Dipartimento di Informatica
Università di Torino
Italy
likavec@di.unito.it

(Received 05 10 2010)
(Revised 19 07 2011)

University of Lyon
École Normal Supérieure de Lyon
France
pierre.lescanne@ens-lyon.fr