Understanding untyped $\overline{\lambda}\mu\widetilde{\mu}$ calculus

Pierre Lescanne, Silvia Likavec

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

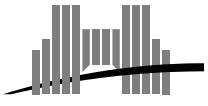
Understanding untyped $\overline{\lambda}\mu\widetilde{\mu}$ calculus

Pierre Lescanne
Silvia Likavec

November 2004

Research Report N° RR2004-50

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`



Understanding untyped $\overline{\lambda}\mu\widetilde{\mu}$ calculus

Pierre Lescanne
Silvia Likavec

November 2004

Abstract

We prove the confluence of $\overline{\lambda}\mu\widetilde{\mu}_T$ and $\overline{\lambda}\mu\widetilde{\mu}_Q$, two well-behaved subcalculi of the $\overline{\lambda}\mu\widetilde{\mu}$ calculus, closed under call-by-name and call-by-value reduction, respectively. Moreover, we give the interpretation of $\overline{\lambda}\mu\widetilde{\mu}_T$ in the category of negated domains, and the interpretation of $\overline{\lambda}\mu\widetilde{\mu}_Q$ in the Kleisli category. To the best of our knowledge this is the first interpretation of *untyped* $\overline{\lambda}\mu\widetilde{\mu}$ calculus.

Keywords: Continuation semantics, classical logic, categories, type theory

Résumé

On prouve la confluence de $\overline{\lambda}\mu\widetilde{\mu}_T$ et $\overline{\lambda}\mu\widetilde{\mu}_Q$, deux sous-calculs de $\overline{\lambda}\mu\widetilde{\mu}$ dotés de bonnes propriétés et clos par réduction en appel par nom et en appel par valeur, respectivement. De plus, on donne l'interprétation de $\overline{\lambda}\mu\widetilde{\mu}_T$ dans la catégorie des "domaines niés" et l'interprétation de $\overline{\lambda}\mu\widetilde{\mu}_Q$ dans la catégorie de Kleisli. A notre connaissance, cela constitue la première interpétation non typée du $\overline{\lambda}\mu\widetilde{\mu}$ calcul.

Mots-clés: sémantique par continuation, logique classique, catégories, théorie des types

1 Introduction

When interpreting 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 [21] in order to formalize flow control in programming languages. Continuation-passing-style (cps) translations were introduced by Fisher and Reynolds in [6] and [17] for call-by-value lambda calculus, whereas a call-by-name variant was introduced by Plotkin in [16]. Moggi gave a semantic version of call-by-value cps translation in his study of notions of computation in [14]. Lafont [10] introduced cps translation of call-by-name $\lambda\mathcal{C}$ calculus [4], [5] to a fragment of lambda calculus that corresponds to the $\neg, \wedge$-fragment of intuistionistic logic.

Categorical semantics for both call-by-name and call-by-value versions of Parigot's $\lambda\mu$ calculus [15] with disjunction types was given by Selinger in [20]. The two variants of $\lambda\mu$ calculus are shown to be isomorphic in the presence of product and disjunction types. Hofmann and Streicher presented categorical continuation models for call-by-name $\lambda\mu$ calculus in [9] and showed the completeness. Lengrand gave categorical semantics for typed $\overline{\lambda}\mu\widetilde{\mu}$ calculus and $\lambda\xi$ calculus (implicational fragment of the classical sequent calculus LK) in [13]. First attempt to give denotational semantics for pure (untyped) $\lambda\mu$ calculus is presented in Laurent [12] by defining a type system with intersection and union types.

Although the original $\overline{\lambda}\mu\widetilde{\mu}$ calculus of [2] has a system of simple types based on the sequent calculus, the untyped version is a Turing-complete language for computation with explicit representation of control, as well as code. In this work we try to give a meaning to *untyped* $\overline{\lambda}\mu\widetilde{\mu}$ calculus and understand its behaviour. We interpret its variant closed under call-by-name reduction in the category of negated domains, and the variant closed under call-by-value reduction in the Kleisli category. As far as we know, this is the first interpretation of *untyped* $\overline{\lambda}\mu\widetilde{\mu}$ calculus. We also prove the confluence of both versions.

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 $\overline{\lambda}\mu\widetilde{\mu}_T$ and $\overline{\lambda}\mu\widetilde{\mu}_Q$. Section 4 gives an account of negated categories where we interpret $\overline{\lambda}\mu\widetilde{\mu}_T$ calculus. In Section 5 we present the basic notions of Kleisli triple and Kleisli category and interpret $\overline{\lambda}\mu\widetilde{\mu}_Q$ calculus. Finally, we give the improved interpretation of $\overline{\lambda}\mu\widetilde{\mu}_T$ calculus. We conclude in Section 6.

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

2.1 Intuition and syntax

$\overline{\lambda}\mu\widetilde{\mu}$ calculus was introduced by Curien and Herbelin in [2], giving a Curry-Howard correspondence for classical logic. The terms of $\overline{\lambda}\mu\widetilde{\mu}$ represent derivations in a sequent calculus proof system and reduction reflects the process of cut-elimination.

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 Dougherty, Ghilezan and Lescanne in [7] and [3].

The basic syntactic entities are 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:

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

There are two kinds of variables in the calculus: the set Var_v of *caller variables* (denoted by $x, y, \ldots$) and the set Var_e, of *callee variables* (denoted by $\alpha, \beta, \ldots$). The caller variables can be bound by λ abstraction and 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.

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 take place as one of its internal callee variables. A callee can ask a caller to take place as one of its internal caller variables. The components can be nested and more processes can be active at the same time.

In [2], the basic constructs are called *commands, terms*, and *contexts*. The present names for the syntactic constructs of the calculus were chosen by Ghilezan and Lescanne in [7], since they reflect better the symmetry of the calculus. Also, it should be possible to use the notion "term" to refer to all the expressions of the calculus, not just to a subset of terms. Finally, commands definitely do not denote commands. We use this new terminology in our work.

2.2 Reduction rules

There are only three rules that characterise the reduction in $\overline{\lambda}\mu\widetilde{\mu}$:

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

The above substitutions are defined as to avoid variable capture [1].

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 ambiguously, producing two different results. For example,

$$\begin{array}{l} \langle \mu\alpha.\langle y \parallel \beta\rangle \parallel \widetilde{\mu}x.\langle z \parallel \gamma\rangle\rangle \rightarrow_{\mu} \langle y \parallel \beta\rangle \text{ and} \\ \langle \mu\alpha.\langle y \parallel \beta\rangle \parallel \widetilde{\mu}x.\langle z \parallel \gamma\rangle\rangle \rightarrow_{\widetilde{\mu}} \langle z \parallel \gamma\rangle. \end{array}$$

Hence, the calculus is not confluent. But if the priority is given to one of the rules, we obtain two confluent subcalculi $\overline{\lambda}\mu\widetilde{\mu}_T$ and $\overline{\lambda}\mu\widetilde{\mu}_Q$. We give the details in the next section.

2.3 Two confluent subcalculi

There are two possible reduction strategies in the calculus that depend on the orientation of the critical pair, [2]. If the priority is given to $(\widetilde{\mu})$ redexes, we obtain the calculus $\overline{\lambda}\mu\widetilde{\mu}_T$, closed under call-by-name reduction, whereas giving the priority to (μ) redexes, we obtain $\overline{\lambda}\mu\widetilde{\mu}_Q$ calculus with call-by-value reduction strategy.

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

$\overline{\lambda}\mu\widetilde{\mu}_T$	$\overline{\lambda}\mu\widetilde{\mu}_Q$
$c ::= \langle v \parallel e\rangle$	$c ::= \langle v \parallel e\rangle$
$E ::= \alpha \mid v \bullet E$	$V ::= x \mid \lambda x.v$
$v ::= x \mid \lambda x.v \mid \mu\alpha.c$	$e ::= \alpha \mid \widetilde{\mu}x.c \mid V \bullet e$
$e ::= E \mid \widetilde{\mu}x.c$	$v ::= V \mid \mu\alpha.c$

In $\overline{\lambda}\mu\widetilde{\mu}_T$ we distinguish a subset E of callees, called *applicative contexts*. In $\overline{\lambda}\mu\widetilde{\mu}_Q$, notice the presence of *values* V, which form the subset of the set of callers and help distinguish values from the rest of computations.

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

$$\begin{array}{ll} (\rightarrow) & \langle \lambda x.v_1 \parallel v_2 \bullet E\rangle \rightarrow \langle v_1[x \leftarrow v_2] \parallel E\rangle \\ (\mu) & \langle \mu\alpha.c \parallel E\rangle \rightarrow c[\alpha \leftarrow E] \\ (\widetilde{\mu}) & \langle v \parallel \widetilde{\mu}x.c\rangle \rightarrow c[x \leftarrow v] \end{array}$$

$$\begin{array}{ll} (\rightarrow') & \langle \lambda x.v_1 \parallel V_2 \bullet e\rangle \rightarrow \langle V_2 \parallel \widetilde{\mu}x.\langle v_1 \parallel e\rangle\rangle \\ (\mu) & \langle \mu\alpha.c \parallel e\rangle \rightarrow c[\alpha \leftarrow e] \\ (\widetilde{\mu}) & \langle V \parallel \widetilde{\mu}x.c\rangle \rightarrow c[x \leftarrow V] \end{array}$$

Let us notice that in $\overline{\lambda}\mu\widetilde{\mu}_T$, we are allowed to consider $(\rightarrow)$ reduction, since $(\rightarrow')$ rule can be immediately followed by the $(\widetilde{\mu})$ rule, which has the priority. On the other hand, in $\overline{\lambda}\mu\widetilde{\mu}_Q$, we

have to use the rule $(\rightarrow')$, since the priority is given to (μ) rule. A different choice would be to consider only the $(\rightarrow')$ rule for both subcalculi, but we think this 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 $\overline{\lambda}\mu\widetilde{\mu}$, we first prove the confluence for each of them. We adopt the technique of parallel reductions given by Takahashi in [24]. 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 $\overline{\lambda}\mu\widetilde{\mu}_T$. We denote the union of the three reduction relations for $\overline{\lambda}\mu\widetilde{\mu}_T$ by $\rightarrow_n$ and its reflexive and transitive closure by $\twoheadrightarrow_n$.

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

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

Definition 3.1 *[**Parallel reduction for** $\overline{\lambda}\mu\widetilde{\mu}_T$]*
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 3.2

1. *For every term G, $G \Rightarrow_n G$.*
2. *If $G \rightarrow_n G'$ then $G \Rightarrow_n G'$*
3. *If $G \Rightarrow_n G'$ then $G \twoheadrightarrow_n G'$*
4. *If $G \Rightarrow_n G'$, $H \Rightarrow_n H'$, then $G[x \leftarrow H] \Rightarrow_n G'[x \leftarrow H']$.*

Proof: See Appendix.

From 2. and 3. we conclude $\twoheadrightarrow_n$ is the reflexive and transitive closure of $\Rightarrow_n$.

3.2 Confluence of $\overline{\lambda}\mu\widetilde{\mu}_T$

Next, we define the term G^* obtained from G by simultaneously reducing all the existing redexes of the term G.

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

$(*1_n)$ $x^* \equiv x$ $\quad$ $(*2_n)$ $(\lambda x.v)^* \equiv \lambda x.v^*$

$(*3_n)$ $(\mu\alpha.c)^* \equiv \mu\alpha.c^*$ $\quad$ $(*4_n)$ $\alpha^* \equiv \alpha$

$(*5_n)$ $(v \bullet E)^* \equiv v^* \bullet E^*$ $\quad$ $(*6_n)$ $(\widetilde{\mu}x.c)^* \equiv \widetilde{\mu}x.c^*$

$(*7_n)$ $(\langle v \parallel e\rangle)^* \equiv \langle v^* \parallel e^*\rangle$ *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 3.4 (Star property for $\Rightarrow_n$) *If $G \Rightarrow_n G'$ then $G' \Rightarrow_n G^*$.*

Proof: See Appendix.

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

Theorem 3.5 (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 3.5, it follows that $\overline{\lambda}\mu\widetilde{\mu}_T$ is confluent.

Theorem 3.6 (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 Category of negated domains

Category of negated domains $\mathcal{N}_R$ was introduced by Lafont in [10], and can be seen as a specialization of Hofmann and Steicher's category of continuations [8].

If $\mathcal{C}$ is a category with distributive finite products and sums, with a fixed object $R \in \mathcal{C}$ such that exponentials of the form R^A exist for all A, $R^A \neq R^B$ for all $A \neq B$, and $\mathcal{C}$ satisfies mono requirement[1], then such a category $\mathcal{C}$ is called **response category** and R is called **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 **category of continuations** and is denoted by $R^{\mathcal{C}}$. This category is cartesian closed [11] and has a canonical premonoidal structure [20]. This can be summarized as follows:

$$1 \cong R^0 \qquad R^A \times R^B \cong R^{A+B} \qquad (R^B)^{(R^A)} \cong R^{R^A \times B}$$
$$\bot := R^1 \cong R \qquad R^A \parr R^B := R^{A \times B}.$$

Next, let $\mathcal{P}$ be a category of predomains[2] and continuous functions, $\mathcal{D}$ be a category of domains and continuous functions and let R be some fixed domain with bottom $\bot_R$. We will call R a **domain of responses**. For each predomain $A \in \mathcal{P}$ we can form an exponential $R^A \in \mathcal{D}$. Since by assumption, R has a bottom, all the exponentials have bottom elements, given by $\bot_{R^A} = \lambda x : A.\bot_R$ for any $A \in \mathcal{P}$. Then the **category of negated domains** $\mathcal{N}_R$ is a full subcategory of $\mathcal{D}$, where the morphisms operate on exponentials of the form R^A. The category $\mathcal{N}_R$ is actually obtained from the category of continuations just taking the category $\mathcal{P}$ of predomains as a basic category, since it has distributive finite products and sums, and exponentials of the form R^A exist.

Since the category $\mathcal{N}_R$ is cartesian closed and has a least fixpoint operator, for any domain R (see [22]), it has enough structure to interpret functional calculi, especially the calculi with control operators.

[1] the morphism $\partial_A : A \to R^{R^A}$ is monic for all $A \in \mathcal{C}$

[2] **Predomain** is a partial order where all directed subsets have a supremum. It does not necessarily have the least element.

4.2 From ordinary models to continuation models

For the extensional lambda calculus, a model is given by an object C in cartesian closed category, such that C is isomorphic to its function space i.e. $C \cong [C \to C] = C^C$ (see [18], [19]). We call such an object **reflexive object**.

In order to obtain a model of lambda calculus and its extensions in $\mathcal{N}_R$, we have the same requirement in the category $\mathcal{N}_R$, which means that we are looking for an object K such that $K = R^K \times K$ in $\mathcal{D}$. For K which is 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 $C = C^C$ in $\mathcal{D}$ and is called **continuation model** of untyped lambda calculus.

Untyped λ-calculus can be interpreted in $R^K \in \mathcal{N}_R$ [22], and this interpretation can be extended to Felleisen's $\lambda\mathcal{C}$ calculus [5] and untyped version of Parigot's $\lambda\mu$ calculus.

4.3 Semantics for $\overline{\lambda}\mu\widetilde{\mu}_T$

As we have seen, the category $\mathcal{N}_R$ of negated domains is convenient for defining the semantics of the various calculi with control operators, since it allows to explicitly deal with continuations. Therefore, we think it was a good starting point in our quest for better understanding the meaning and behaviour of $\overline{\lambda}\mu\widetilde{\mu}$.

As we have already mentioned, $\overline{\lambda}\mu\widetilde{\mu}$ is not confluent due to the presence of the critical pair $\langle \mu\alpha.c_1 \parallel \widetilde{\mu}x.c_2 \rangle$. Hence, we will consider separately two well-behaved subsyntaxes which are closed either under call-by-name ($\overline{\lambda}\mu\widetilde{\mu}_T$) or under call-by-value reduction ($\overline{\lambda}\mu\widetilde{\mu}_Q$).

Let us now turn to the interpretation of call-by-name variant of untyped $\overline{\lambda}\mu\widetilde{\mu}$ calculus, in the category of negated domains introduced in the previous section.

We define the interpretation functions for all syntactic categories of $\overline{\lambda}\mu\widetilde{\mu}_T$ in the category $\mathcal{N}_R$ of negated domains.

Definition 4.1 *Let K be an initial solution of domain equation $K = R^K \times K$ and let $C = R^K$. With Env we denote the set of environments that map caller variables to elements of C and callee variables to elements of K, i.e. for $\rho \in \mathsf{Env}$, $\forall x \in \mathrm{Var}_v, \rho(x) \in C$ and $\forall \alpha \in \mathrm{Var}_e, \rho(\alpha) \in K$. Then the interpretation functions*

$$\begin{array}{ll} [\![-]\!]_C : \mathsf{CalleR} & \to \mathsf{Env} \to C = R^K \\ [\![-]\!]_K : \mathsf{CalleE} & \to \mathsf{Env} \to K \\ [\![-]\!]_R : \mathsf{Capsule} & \to \mathsf{Env} \to R \end{array}$$

are defined as follows

$$\begin{array}{rcl} \mathsf{CalleR:} & & \\ [\![x]\!]_C\rho & = & \lambda\langle s,k\rangle.s\langle\rho(x),k\rangle \\ [\![\lambda x.v]\!]_C\rho & = & \lambda\langle s,k\rangle.s\langle\lambda\langle s_1,k_1\rangle.[\![v]\!]_C\rho[x := s_1]k_1,k\rangle \\ [\![\mu\alpha.c]\!]_C\rho & = & \lambda\langle s,k\rangle.s\langle\lambda h.[\![c]\!]_R\rho[\alpha := h],k\rangle \\ \mathsf{CalleE:} & & \\ [\![\alpha]\!]_K\rho & = & \langle\lambda\langle s,k\rangle.s\rho(\alpha),\mathsf{stop}\rangle \\ [\![v \bullet E]\!]_K\rho & = & \langle\lambda\langle s,k\rangle.[\![v]\!]_C\rho\langle s,[\![E]\!]_K\rho\rangle,\mathsf{stop}\rangle \\ [\![\widetilde{\mu}x.c]\!]_K\rho & = & \langle\lambda\langle s,k\rangle.[\![c]\!]_R\rho[x := s],\mathsf{stop}\rangle \\ \mathsf{Capsule:} & & \\ [\![\langle v \parallel e\rangle]\!]_R\rho & = & [\![v]\!]_C\rho([\![e]\!]_K\rho) \end{array}$$

We will omit the subscripts in various interpretations, since they can be deduced from the terms being interpreted.

Intuitively, the callers represent *computations* and are mapped into C. Callees represent *continuations* and are mapped into K. Finally, capsules can be seen as *responses*, hence are mapped into R. The distinguished continuation stop represents the *stable state*. Since it does not influence the computation, we can take *any* continuation for stop but this choice (taken from [22]) is justified by the fact that it also works for the simplest continuation model where $C = \Sigma$. $\Sigma = \{\bot, \top\}$ is known

as Sierpinski space, where it is only possible to observe termination (represented by $\bot$) and divergence (represented by $\top$). It is the greatest element of K and is defined as $\mathsf{stop} = \langle \lambda k.\top_R, \mathsf{stop}\rangle$, where $k \in K$ and $\top_R$ is the greatest element of the domain R.

Let us now give some explanations for the given interpretations. First of all, 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 lists of denotations which correspond to denotational versions of 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 data from a callee or ask it to take place of one of its internal callee variables. Hence, callers expect callees as arguments. The double abstraction over callees comes from the necessity to trigger the computation in $(\widetilde{\mu})$ rule. It actually enables applying the current evaluation context to a computation and continuing the computation.

Since a callee can ask a caller to take the place of one of its internal caller variables, it has a functional part that can be applied to a caller (the first part of the pair). The second component of the pair is stop, since during the computation, a new evaluation context is provided by the caller.

Finally, in the case of capsules, the interpretation of the caller is applied to the interpretation of the callee, thus producing an element in R.

Next, we give two lemmas, that will be used in order to prove that the semantics is preserved by the reduction rules.

Lemma 4.2 (Substitution lemma 1) *Let G be the term of $\overline{\lambda}\mu\widetilde{\mu}_T$ calculus (caller, callee, or capsule). Then*

1. $[\![G[x \leftarrow y]]\!]\rho = [\![G]\!]\rho[x := \rho(y)]$;
2. $[\![G[x \leftarrow \lambda y.v]]\!]\rho = [\![G]\!]\rho[x := \lambda\langle s, k\rangle.[\![v]\!]\rho[y := s]k]$;
3. $[\![G[x \leftarrow \mu\alpha.c]]\!]\rho = [\![G]\!]\rho[x := \lambda h.[\![c]\!]\rho[\alpha := h]]$.

Lemma 4.3 (Substitution lemma 2) *Let G be the term of $\overline{\lambda}\mu\widetilde{\mu}_T$ calculus (caller, callee, or capsule). Then*

1. $[\![G[\alpha \leftarrow \beta]]\!]\rho = [\![G]\!]\rho[\alpha := \rho(\beta)]$;
2. $[\![G[\alpha \leftarrow y \bullet E]]\!]\rho = [\![G]\!]\rho[\alpha := \langle \rho(y), [\![E]\!]\rangle]$.
3. $[\![G[\alpha \leftarrow \lambda y.v \bullet E]]\!]\rho = [\![G]\!]\rho[\alpha := \langle \lambda\langle s, k\rangle.[\![v]\!]\rho[y := s]k, [\![E]\!]\rangle]$.
4. $[\![G[\alpha \leftarrow \mu\beta.c \bullet E]]\!]\rho = [\![G]\!]\rho[\alpha := \langle \lambda h.[\![c]\!]\rho[\beta := h], [\![E]\!]\rangle]$.

Proofs of both lemmas are by induction on the structure of G.

Finally, we can prove the following theorem.

Theorem 4.4 (Preservation of semantics for $\overline{\lambda}\mu\widetilde{\mu}_T$) *If $G_1 \to G_2$ then $[\![G_1]\!] = [\![G_2]\!]$.*

Proof: See Appendix.

5 Kleisli category and continuation semantics

5.1 Kleisli category

Kleisli categories provide a categorical semantics of computations based on monads. Since every monad corresponds to Kleisli triple, the semantics can be given based on Kleisli triples that are easier to justify computationally.

When interpreting a programming language in call-by-value setting in a category $\mathcal{C}$, we need to distinguish the objects A that represent values of type A from the objects TA that represent computations of type A. Computations of type A are obtained by applying a functor T (called

notion of computation in [14]) to A. There are certain conditions that T has to satisfy and it turns out that T needs to be 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 computation [14]

Definition 5.1 *A **Kleisli triple** over a category $\mathcal{C}$ is a triple $(T, \eta, \_^*)$, such that for*
- $T : Obj(\mathcal{C}) \to Obj(\mathcal{C})$
- $\eta_A : A \to TA$ *for* $A \in Obj(\mathcal{C})$
- $f^* : TA \to TB$ *for* $f : A \to TB$

the following equations hold:
- $\eta_A^* = id_{TA}$;
- $f^* \circ \eta_A = f$ *for* $f : A \to TB$;
- $g^* \circ f^* = (g^* \circ f)^*$ *for* $f : A \to TB$ *and* $g : B \to TK$.

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

Definition 5.2 *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 objects of $\mathcal{C}$;*
- $\mathcal{C}_T(A, B) = \mathcal{C}(A, TB)$;
- $id_{\mathcal{C}_T} = \eta_A : A \to TA$;
- $g \circ_{\mathcal{C}_T} f = g^* \circ f : A \to TK$ *for* $f \in \mathcal{C}_T(A, B)$ *and* $g \in \mathcal{C}_T(B, K)$.

5.2 Kleisli triple of continuations

Depending on the specific computation we want to model, different Kleisli triples can be chosen. In this work we consider **Kleisli triple of continuations** given by the functor

$$TA = R^{R^A},$$

where R is the fixed object of responses (predomain with a least element and at least one more element), together with the functors
- $\eta_A(a) = \lambda k : R^A.k(a)$ and
- $f^*(s) = \lambda k : R^B.s(\lambda a : A.f(a)(k))$ for $f : A \to TB$ and $s \in TA$.

We denote by $\mathcal{K}_R$ the Kleisli category over the category $\mathcal{P}$ of predomains for a given Kleisli triple of continuations $(T, \eta, \_^*)$. 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 [23], 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.

In the next section, we will see how call-by-value variant of untyped $\overline{\lambda}\mu\widetilde{\mu}$ calculus, can be interpreted in the Kleisli category $\mathcal{K}_R$.

5.3 Semantics for $\overline{\lambda}\mu\widetilde{\mu}_Q$

In this section we will consider $\overline{\lambda}\mu\widetilde{\mu}_Q$, which is a variant of untyped $\overline{\lambda}\mu\widetilde{\mu}$ calculus closed under call-by-value reduction. It permits giving always precedence to (μ) rule.

We give the definition of interpretation functions for all *four* syntactic categories of the calculus. It means that there is an interpretation function also for values, since it prevents the values and computations to be confused. Functions are applied to values, but can still have computations for a result, hence it is necessary to have $W = C^W$.

Definition 5.3 *Let us consider an initial solution of the system of domain equations*

$$W = C^W \quad K = R^W \quad C = R^K.$$

Let Env *be the set of environments that map caller variables to elements of W and callee variables to*

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

$$\begin{array}{ll} [\![-]\!]_W : \mathsf{Value} & \to \mathsf{Env} \to W = C^W \\ [\![-]\!]_K : \mathsf{CalleE} & \to \mathsf{Env} \to K = R^W \\ [\![-]\!]_C : \mathsf{CalleR} & \to \mathsf{Env} \to C = R^K \\ [\![-]\!]_R : \mathsf{Capsule} & \to \mathsf{Env} \to R \end{array}$$

are defined as follows

$$\begin{array}{rcl} \mathsf{Value:} & & \\ [\![x]\!]_W \rho & = & \rho(x) \\ [\![\lambda x.v]\!]_W \rho & = & \lambda w.[\![v]\!]_C \rho[x := w] \\ \mathsf{CalleE:} & & \\ [\![\alpha]\!]_K \rho & = & \rho(\alpha) \\ [\![V \bullet e]\!]_K \rho & = & \lambda w.(w([\![V]\!]_W \rho))([\![e]\!]_K \rho) \\ [\![\widetilde{\mu} x.c]\!]_K \rho & = & \lambda w.[\![c]\!]_R \rho[x := w] \\ \mathsf{CalleR:} & & \\ [\![x]\!]_C \rho & = & \lambda k.k[\![x]\!]_W \rho \\ [\![\lambda x.v]\!]_C \rho & = & \lambda k.k[\![\lambda x.v]\!]_W \rho \\ [\![\mu \alpha.c]\!]_C \rho & = & \lambda k.[\![c]\!]_R \rho[\alpha := k] \\ \mathsf{Capsule:} & & \\ [\![\langle v \parallel e \rangle]\!]_R \rho & = & [\![v]\!]_C \rho([\![e]\!]_K \rho) \end{array}$$

We will leave the subscript only in the case of $[\![-]\!]_W$ to avoid the ambiguity.

One important difference when interpreting call-by-value calculus is that variables are interpreted as values, i.e. $\rho(x) \in W$, whereas in call-by-name case values 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 elements of W, callers as computations are elements of $C = R^{R^W}$, callees as continuations are elements of $K = R^W$, and capsules as responses are elements of R.

Also notice, that the interpretation of values in C is obtained by applying the function $\eta_A(a) = \lambda k : R^A.k(a)$ from a Kleisli triple, to the interpretation of values in W. Hence, we include values into computations. On the other hand, $\mu\alpha.c$ is not a value, hence its interpretation is given only in C.

In the case of callees, $V \bullet e$ and $\widetilde{\mu} x.c$ can be seen as call-by-value evaluation contexts. Hence, for $V \bullet e$ the computation (seen as value) is applied to V and then evaluated in the evaluation context e. For $\widetilde{\mu} x.c$, the caller is just fed into a capsule c.

As in the previous section, we first give some lemmas that will be used later to prove the preservation of semantics under reduction.

Lemma 5.4 (Substitution lemma 1) *Let G be the term of $\overline{\lambda}\mu\widetilde{\mu}_Q$ calculus (caller, callee, or capsule). Then*

1. $[\![G[x \leftarrow y]]\!]_{(W)} \rho = [\![G]\!]_{(W)} \rho[x := \rho(y)]$;
2. $[\![G[x \leftarrow \lambda y.v]]\!]_{(W)} \rho = [\![G]\!]_{(W)} \rho[x := [\![\lambda y.v]\!]_W \rho]$;

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

Proof: By induction on the structure of G.

Lemma 5.5 (Substitution lemma 2) *Let G be the term of $\overline{\lambda}\mu\widetilde{\mu}_Q$ calculus (caller, callee, or capsule). Then $[\![G[\alpha \leftarrow e]]\!] \rho = [\![G]\!] \rho[\alpha := [\![e]\!] \rho]$*

Proof: By induction on the structure of G followed by induction on the structure of e.

Theorem 5.6 (Preservation of semantics for $\overline{\lambda}\mu\widetilde{\mu}_Q$) *If $G_1 \to G_2$ then $[\![G_1]\!] = [\![G_2]\!]$.*

Proof: See Appendix.

5.4 Improving the semantics for $\overline{\lambda}\mu\widetilde{\mu}_T$

In this part, we would still keep the domain equations from Section 4.3, but we will also try to integrate the ideas from Section 5.3.

In Section 5.3, we considered two different types of computations, namely *values* as elements of W and *computations* as elements of C. With the help of the functor $\eta_A(a) = \lambda k : R^A.k(a)$ from Kleisli triple, we had a way of including values into 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 [2]), from the rest of continuations. Co-values will be interpreted in K as in Section 4.3, but callees will be interpreted in R^{R^K}. The functor

$$\eta_K(k) = \lambda s : R^{R^K}.s(k)$$

from the Kleisli triple will serve to include co-values into continuations.

We give interpretation functions for all the *four* syntactic constructs of $\overline{\lambda}\mu\widetilde{\mu}_T$, which means that the interpretation function is also given for co-values, thus making a clear difference between them and rest of callees.

Definition 5.7 *Let K be an initial solution of domain equation $K = R^K \times K$ and let $C = R^K$ and $F = R^C$. With* Env *we denote the set of environments that map caller variables to elements of C and callee variables to elements of K, i.e. for $\rho \in$ Env, $\forall x \in \mathrm{Var}_v, \rho(x) \in C$ and $\forall \alpha \in \mathrm{Var}_e, \rho(\alpha) \in K$. Then the interpretation functions*

$$\begin{array}{ll} [\![-]\!]_K : \textit{Co-value} & \to \textit{Env} \to K \\ [\![-]\!]_C : \textit{CalleR} & \to \textit{Env} \to C = R^K \\ [\![-]\!]_F : \textit{CalleE} & \to \textit{Env} \to F = R^C \\ [\![-]\!]_R : \textit{Capsule} & \to \textit{Env} \to R \end{array}$$

are defined as follows

$$\begin{array}{rcl} \textit{Co-value:} & & \\ [\![\alpha]\!]_K\rho & = & \rho(\alpha) \\ [\![v \bullet E]\!]_K\rho & = & \langle [\![v]\!]_C\rho, [\![E]\!]_F\rho \rangle \\ \textit{CalleR:} & & \\ [\![x]\!]_C\rho & = & \rho(x) \\ [\![\lambda x.v]\!]_C\rho & = & \lambda\langle s,k\rangle.[\![v]\!]_C\rho[x := s]k \\ [\![\mu\alpha.c]\!]_C\rho & = & \lambda k.[\![c]\!]_R\rho[\alpha := k] \\ \textit{CalleE:} & & \\ [\![\alpha]\!]_F\rho & = & \lambda s.s([\![\alpha]\!]_K\rho) \\ [\![v \bullet E]\!]_F\rho & = & \lambda s.s([\![v \bullet E]\!]_K\rho) \\ [\![\widetilde{\mu}x.c]\!]_F\rho & = & \lambda s.[\![c]\!]_R\rho[x := s] \\ \textit{Capsule:} & & \\ [\![\langle v \parallel e\rangle]\!]_R\rho & = & [\![e]\!]_F\rho([\![v]\!]_C\rho) \end{array}$$

We leave the subscript only in the case of $[\![-]\!]_F$ to avoid the ambiguity.

We can see the different syntactic constructs of $\overline{\lambda}\mu\widetilde{\mu}_T$ as elements of the following semantical objects: callers as computations are elements of $C = R^K$, co-values as basic continuations are elements of $K \cong R^K \times K$, callees as continuations are elements of $F = R^C$, and capsules as responses are elements of R.

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

We can again prove the Substitution lemma and Preservation of semantics under reduction.

Lemma 5.8 (Substitution lemma) *Let G be the term of $\overline{\lambda}\mu\widetilde{\mu}_T$ calculus (caller, callee, or capsule). Then*

1. $[\![G[x \leftarrow v]]\!]\rho = [\![G]\!]\rho[x := [\![v]\!]\rho]$;

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

where $[\![-]\!]_{(K)}$ *means that lemma holds for both interpretations,* $[\![-]\!]_F$ *and* $[\![-]\!]_K$.

Theorem 5.9 (Preservation of semantics for $\overline{\lambda}\mu\widetilde{\mu}_T$) *If* $G_1 \to G_2$ *then* $[\![G_1]\!] = [\![G_2]\!]$.

Proof: See Appendix.

6 Conclusions and future work

As a step towards better understanding of denotational semantics of $\overline{\lambda}\mu\widetilde{\mu}$ calculus, we interpreted its *untyped* call-by-name ($\overline{\lambda}\mu\widetilde{\mu}_T$) and call-by-value ($\overline{\lambda}\mu\widetilde{\mu}_Q$) versions, which permits us to exploit $\overline{\lambda}\mu\widetilde{\mu}$ as a programming language. Continuation semantics of $\overline{\lambda}\mu\widetilde{\mu}_T$ is given by the interpretation in the category of negated domains of [22], whereas $\overline{\lambda}\mu\widetilde{\mu}_Q$ is interpreted in Moggi's Kleisli category over predomains for the continuation monad [14]. Using computational monads, we also give an improved interpretation for $\overline{\lambda}\mu\widetilde{\mu}_T$. As a first research direction it would be interesting to better understand the correspondence between these interpretations.

Another important contribution of this work is the proof of confluence for both versions of $\overline{\lambda}\mu\widetilde{\mu}$.

We would like to extend the present work to the complete symmetric calculus of [2] and find the interpretation for all the constructs of that calculus, including $e \bullet v$ and $\beta\lambda.e$. It seems that this is not a trivial task, and that we need better understanding of the behaviour of these terms.

Another interesting direction to follow would be to interpret the typed $\overline{\lambda}\mu\widetilde{\mu}$ calculus in Selinger's control (co-control) categories [20], for both call-by-name and call-by-value variant of the calculus, similar to the ones given in [13], but giving different interpretations for types (closer to the ones given in [8]).

Finally, thorough analysis of semantics of the calculus typed using intersection and union types [3] is foreseen.

References

[1] H. P. Barendregt. *The Lambda Calculus: its Syntax and Semantics*. North-Holland, Amsterdam, revised edition, 1984.

[2] P.-L. Curien and H. Herbelin. The Duality of Computation. In *Proc. 5th ACM SIGPLAN International Conference on Functional Programming (ICFC'00)*, pages 233–243. ACM Press, 2000.

[3] D. Dougherty, S. Ghilezan, and P. Lescanne. Characterizing strong normalization in a language with control operators. In *Proc. 6th ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming PPDP'04*, 2004.

[4] M. Felleisen, D. P. Friedman, E. Kohlbecker, and B. F. Duba. A syntactic theory of sequential control. *Theoretical Computer Science*, 52(3):205–237, 1987.

[5] M. Felleisen and R. Hieb. The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 103(2):235–271, 1992.

[6] M. Fischer. Lambda calculus schemata. In *Proc. ACM Conference on Proving Assertions About Programs '72*, pages 104–109. ACM Press, 1972.

[7] S. Ghilezan and P. Lescanne. Classical proofs, typed processes and intersection types. In *International Workshop TYPES'03 (Selected Papers)*, volume 3085 of *LNCS*, pages 226–241. Springer-Verlag, 2004.

[8] 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*, pages 387–397. IEEE Computer Society Press, 1997.

[9] M. Hofmann and Th. Streicher. Completeness of continuation models for $\lambda\mu$-calculus. *Information and Computation*, 179(2):332 – 355, 2002.

[10] Y. Lafont. Negation versus implication. Draft, 1991.

[11] Y. Lafont, B. Reus, and Th. Streicher. Continuation semantics or expressing implication by negation. Technical Report 93-21. University of Munich, 1993.

[12] O. Laurent. On the denotational semantics of the pure lambda-mu calculus. Manuscript, 2004.

[13] S. Lengrand. Call-by-value, call-by-name, and strong normalization for the classical sequent calculus. In B. Gramlich and S. Lucas, editors, *Electronic Notes in Theoretical Computer Science*, volume 86. Elsevier, 2003.

[14] E. Moggi. Notions of computations and monads. *Information and Computation*, 93(1), 1991.

[15] M. Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In *Proc. International Conference on Logic Programming and Automated Reasoning, LPAR'92*, volume 624 of *LNCS*, pages 190–201. Springer Verlag, 1992.

[16] G. D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[17] J. C. Reynolds. Definitional interpreters for higher-order programming languages. In *Proc. ACM Annual Conference*, pages 717–740. ACM Press, 1972.

[18] D. S. Scott. Continuous lattices. In F.W.Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, volume 274 of *Lecture Notes in Mathematics*, pages 97–136. Springer-Verlag, 1972.

[19] D. S. Scott. Domains for denotational semantics. In M. Nielsen and E. M. Schmidt, editors, *Automata, Languages and Programming*, volume 140 of *Lecture Notes in Computer Science*, pages 577–613. Springer-Verlag, 1982.

[20] P. Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Mathematical Structures in Computer Science*, 11:207–260, 2001.

[21] C. Strachey and C.P. Wadsworth. Continuations: A mathematical semantics for handling full jumps. Oxford University Computing Laboratory Technical Monograph PRG-11, 1974.

[22] Th. Streicher and B. Reus. Classical logic, continuation semantics and abstract machines. *Journal of Functional Programming*, 8(6):543–572, 1998.

[23] Th. Streicher and B. Reus. Continuation semantics: Abstract machines and control operators. Unpublished manuscript, 1998.

[24] M. Takahashi. Parallel reduction in λ-calculus. *Information and Computation*, 118:120–127, 1995.

A Appendix

Proof of Lemma 3.2

1. By induction on the structure of G. 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 $(g2_n), (g3_n), (g5_n) - (g7_n)$). □

2. By induction on the context of the redex. If $G \to_n G'$ then $G = \mathcal{C}[H]$, $G' = \mathcal{C}[H']$ and $H \to_n H'$. We just show a few illustrative cases.

   * If $\mathcal{C} = [\,]$, then $H \to_n H'$ can be one of the following:
     - $H = \langle \lambda x.v_1 \parallel v_2 \bullet E\rangle$ and $H' = \langle v_1[x \leftarrow v_2] \parallel E\rangle$. Then by $(g8_n)$, $H \Rightarrow_n H'$ since $v_i \Rightarrow_n v_i$ $i = 1, 2$ and $E \Rightarrow_n E$ by Lemma 3.2(1).
     - $H = \langle \mu\alpha.c \parallel E\rangle$ 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 3.2(1).
     - $H = \langle v \parallel \widetilde{\mu}x.c\rangle$ and $H' = c[x \leftarrow v]$. Then $H \Rightarrow_n H'$ using $(g10_n)$ and Lemma 3.2(1) since $v \Rightarrow_n v$ and $c \Rightarrow_n c$.

   * If $\mathcal{C} = \widetilde{\mu}x.\mathcal{C}'[\,]$, then $G = \widetilde{\mu}x.\mathcal{C}'[H]$ and $G' = \widetilde{\mu}x.\mathcal{C}'[H']$. By the induction hypothesis, $\mathcal{C}'[H] \Rightarrow_n \mathcal{C}'[H']$, so by $(g3_n)$ of the definition of $\Rightarrow_n$ we get $G \Rightarrow_n G'$.

   * If $\mathcal{C} = \langle \mu\alpha.c \parallel \mathcal{C}'[\,]\rangle$, then $G = \langle \mu\alpha.c \parallel \mathcal{C}'[H]\rangle$ and $G' = \langle \mu\alpha.c \parallel \mathcal{C}'[H']\rangle$. By the induction hypothesis, $\mathcal{C}'[H] \Rightarrow_n \mathcal{C}'[H']$, so by $(g7_n)$ $G = \langle \mu\alpha.c \parallel \mathcal{C}'[H]\rangle \Rightarrow_n \langle \mu\alpha.c \parallel \mathcal{C}'[H']\rangle = G'$.

3. By induction on the structure of G.

4. By induction on definition of $G \Rightarrow_n G'$. □

Proof of Theorem 3.4 By induction on the structure of G. Since all the cases follow by the straightforward induction, we show only a few illustrative ones.

1. If $G = x$, then G can only parallel reduce to x itself and $x \equiv x^*$ which is G^*.

2. When $G = \mu\alpha.c$, then $\mu\alpha.c \Rightarrow_n \mu\alpha.c'$ for some c' such that $c \Rightarrow_n c'$. By the induction hypothesis, $c' \Rightarrow_n c^*$, hence $\mu\alpha.c' \Rightarrow_n \mu\alpha.c^* = G^*$.

4. For $G = \langle \lambda x.v_1 \parallel v_2 \bullet E\rangle$, if $\langle \lambda x.v_1 \parallel v_2 \bullet E\rangle \Rightarrow_n G'$, we distinguish two subcases:

   * $G' = \langle \lambda x.v_1' \parallel v_2' \bullet E'\rangle$ 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, either $\langle \lambda x.v_1' \parallel v_2' \bullet E'\rangle \Rightarrow_n \langle \lambda x.v_1^* \parallel v_2^* \bullet E^*\rangle \Rightarrow_n \langle v_1^*[x \leftarrow v_2^*] \parallel E^*\rangle$ by $(g7_n)$ followed by $(g8_n)$ or $\langle \lambda x.v_1' \parallel v_2' \bullet E'\rangle \Rightarrow_n \langle v_1^*[x \leftarrow v_2^*] \parallel E^*\rangle$ by $(g8_n)$.

   * $G' = \langle v_1'[x \leftarrow v_2'] \parallel E'\rangle$ 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, $\langle v_1'[x \leftarrow v_2'] \parallel E'\rangle \Rightarrow_n \langle v_1^*[x \leftarrow v_2^*] \parallel E^*\rangle$ by Lemma 3.2(4) and $(g7_n)$. □

Proof of Theorem 5.6

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

   $[\![\langle \lambda x.v \parallel V \bullet e\rangle]\!]\rho = [\![\lambda x.v]\!]\rho([\![V \bullet e]\!]\rho)$
   $= (\lambda k.k(\lambda w.[\![v]\!]\rho[x := w]))(\lambda w_1.(w_1([\![V]\!]_W\rho))([\![e]\!]\rho))$
   $= (\lambda w_1.(w_1([\![V]\!]_W\rho))([\![e]\!]\rho))(\lambda w.[\![v]\!]\rho[x := w])$
   $= (\lambda w.[\![v]\!]\rho[x := w])([\![V]\!]_W\rho)([\![e]\!]\rho)$
   $= [\![v]\!]\rho[x := [\![V]\!]_W\rho][\![e]\!]\rho$
   $= [\![v[x \leftarrow V]]\!]\rho([\![e]\!]\rho)$

   $[\![\langle V \parallel \widetilde{\mu}x.\langle v \parallel e\rangle\rangle]\!]\rho = [\![V]\!]\rho([\![\widetilde{\mu}x.\langle v \parallel e\rangle]\!]\rho)$
   $= [\![\langle v \parallel e\rangle[x \leftarrow V]]\!]\rho$ (as in 3.)
   $= [\![v[x \leftarrow V]]\!]\rho([\![e]\!]\rho)$ since $x \notin e$

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

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

3. $\langle V \parallel \widetilde{\mu} x.c\rangle \to c[x \leftarrow V]$

   We proceed by induction on the structure of V.

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

Proof of Theorem 5.9

1. $\langle \lambda x.v_1 \parallel v_2 \bullet E\rangle \to \langle v_1[x \leftarrow v_2] \parallel E\rangle$

   $[\![\langle \lambda x.v_1 \parallel v_2 \bullet E\rangle]\!]\rho = [\![v_2 \bullet E]\!]\rho([\![\lambda x.v_1]\!]\rho)$
   $= (\lambda s.s([\![v_2 \bullet E]\!]_K\rho))([\![\lambda x.v_1]\!]\rho)$
   $= [\![\lambda x.v_1]\!]\rho\langle [\![v_2]\!]\rho, [\![E]\!]_K\rho\rangle$
   $= (\lambda\langle s,k\rangle.[\![v_1]\!]\rho[x := s]k)\langle [\![v_2]\!]\rho, [\![E]\!]_K\rho\rangle$
   $= [\![v_1]\!]\rho[x := [\![v_2]\!]\rho]([\![E]\!]_K\rho) = [\![v_1[x \leftarrow v_2]]\!]\rho([\![E]\!]_K\rho)$

   $[\![\langle v_1[x \leftarrow v_2] \parallel E\rangle]\!]\rho = [\![E]\!]\rho([\![v_1[x \leftarrow v_2]]\!]\rho)$
   $= (\lambda s.s([\![E]\!]_K\rho))([\![v_1[x \leftarrow v_2]]\!]\rho)$
   $= [\![v_1[x \leftarrow v_2]]\!]\rho([\![E]\!]_K\rho)$

2. $\langle \mu\alpha.c \parallel E\rangle \to c[\alpha \leftarrow E]$

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

3. $\langle v \parallel \widetilde{\mu} x.c\rangle \to c[x \leftarrow v]$

   $[\![\langle v \parallel \widetilde{\mu} x.c\rangle]\!]\rho = (\lambda s.[\![c]\!]\rho[x := s])([\![v]\!]\rho)$
   $= [\![c]\!]\rho[x := [\![v]\!]\rho] = [\![c[x \leftarrow v]]\!]\rho$ □