A CPS-Translation of the $\lambda\mu$-Calculus

Philippe de Groote

INRIA-Lorraine – CRIN – CNRS
Campus Scientifique - B.P. 239
54506 Vandœuvre-lès-Nancy Cedex – FRANCE
e-mail: degroote@loria.fr

Abstract. We present a translation of Parigot's $\lambda\mu$-calculus [10] into the usual λ-calculus. This translation, which is based on the so-called *continuation passing style*, is correct with respect to equality and with respect to evaluation. At the type level, it induces a logical interpretation of classical logic into intuitionistic one, akin to Kolmogorov's negative translation. As a by-product, we get the normalization of second order typed $\lambda\mu$-calculus.

1 Introduction

During the last three years, several authors have introduced various systems that clarify the computational content of classical proofs [2, 3, 5, 6, 8, 9, 10]. In this paper, we investigate one of these systems, namely Parigot's $\lambda\mu$-calculus [10].

Our investigation tool is merely syntactic: we propose a translation of the $\lambda\mu$-calculus into the well known λ-calculus. This interpretation, which obey a continuation passing style, works for any $\lambda\mu$-term. It is therefore more general than the one introduced by M. Parigot in [11], which works only for bounded $\lambda\mu$-terms.

The notion of continuation has been developed in the framework of programming language semantics in order to model control. Since Griffin's work [6], one knows that there is a connection between classical proofs and control. With this respect, our interpretation enlighten the relation between the $\lambda\mu$-calculus and the notion of control: a μ-abstraction is interpreted as a λ-abstraction whose bound variable stands for some possible continuation.

The main properties of our translation are that it is correct with respect to equality and evaluation: two $\lambda\mu$-terms are equal if and only if their translations are (*translation property*); the evaluation of a $\lambda\mu$-term may be simulated faithfully by the evaluation of its translation (*simulation property*).

Up to a slight modification, the interpretation of the typed $\lambda\mu$-calculus that results from our translation does also make sense from a proof-theoretic point of view. It amounts to a negative translation of classical logic into intuitionistic one, akin to Kolmogorov's. This allow us to establish the normalization of second-order classical natural deduction, which is a property that has been recently proven in [12].[1]

[1] Actually, M. Parigot proves more, namely strong normalization.

In: S. Tison (Ed.), *Proceedings of the Colloquium on Trees in Algebra and Programming — CAAP'94*, Lecture Notes in Computer Science, Vol. 787 Springer-Verlag (1994), pp. 85–99.

The remainder of this paper is organized as follows. The next section is a short introduction to Parigot's $\lambda\mu$-calculus. In Section 3, we define our translation of the $\lambda\mu$-calculus into the λ-calculus, and we prove its correctness with respect to equality. In Section 4, we address the problem of the correctness of the translation with respect to evaluation. The proof-theoretic interpretation of the translation is investigated in Section 5. Finally we present our conclusions in Section 6.

2 The $\lambda\mu$-Calculus

This section is a short introduction to the $\lambda\mu$-calculus. The reader may refer to [10, 11, 12] for further details.

The $\lambda\mu$-calculus, introduced by M. Parigot in [10], extends the λ-calculus in order to give an algorithmic interpretation to classical proofs. This interpretation is based on cut-elimination as it is in the case of intuitionistic logic. Nevertheless, in addition to the so-called *logical reductions* of intuitionistic logic, some other kind of reduction is needed in order to handle the double-negation rule of classical logic. This gives rise to an extension of the syntax of the λ-calculus (addition of *μ-abstractions* and *named terms*), and to a new notion of reduction (the one of *structural reduction*).

The terms of the $\lambda\mu$-calculus ($\lambda\mu$-terms, for short) are built from two distinct alphabets of variables: the set of λ-variables, and the set μ-variables. The raw syntax of the language is given by the following grammar:

$$T \;::=\; x \mid (\lambda x.\, T) \mid (T\, T) \mid (\mu\delta.\, T) \mid [\delta]\, T,$$

where x ranges over λ-variables, and δ ranges over μ-variables. A $\lambda\mu$-term of the form $\mu\delta.\, T$ is called a *μ-abstraction*, and a $\lambda\mu$-term of the form $[\delta]\, T$ is called a *named term*. The operator μ is a binding operator as is λ. Therefore, the free occurrences of a μ-variable δ in T become bound in $\mu\delta.\, T$. In order to be protected from clashes between free and bound variables, we adopt Barendregt's variable convention [4] for μ-variables as well as for λ-variables.

The reduction relation of the $\lambda\mu$-calculus is induced by three different notions of reduction. The first one is the usual notion of reduction β:

$$(\lambda x.\, M)\, N \to M[x{:=}N]$$

where $M[x{:=}N]$ denotes the usual capture-avoiding substitution.

The second notion of reduction is the one of *structural reduction*. This notion may be intuitively explained as follows: in a $\lambda\mu$-term $\mu\alpha.\, M$ of type $A \to B$, only the subterms named by α are *really* of type $A \to B$ (see the typing rules hereafter); hence, when such a μ-abstraction is applied to an argument, this argument must be passed over to the subterms named by α. This intuition is formalized as follows:

$$(\mu\delta.\, M)\, N \to \mu\delta.\, M[\delta \Leftarrow N],$$

where the structural substitution is inductively defined as follows:

(i) $x[\delta \Leftarrow N] \;=\; x;$

(ii) $(\lambda x.\, M)[\delta \Leftarrow N] \;=\; \lambda x.\, M[\delta \Leftarrow N];$

(iii) $(M\, O)[\delta \Leftarrow N] \;=\; M[\delta \Leftarrow N]\, O[\delta \Leftarrow N];$

(iv) $(\mu\gamma.\, M)[\delta \Leftarrow N] \;=\; \mu\gamma.\, M[\delta \Leftarrow N];$

(v) $([\,\delta\,]\, M)[\delta \Leftarrow N] \;=\; [\,\delta\,]\,(M[\delta \Leftarrow N]\, N);$

(vi) $([\,\gamma\,]\, M)[\delta \Leftarrow N] \;=\; [\,\gamma\,]\, M[\delta \Leftarrow N] \quad \textit{if } \delta \neq \gamma.$

The third notion of reduction, which is called *renaming*, is the following:

$$[\,\delta\,]\,(\mu\gamma.\, M) \to M[\gamma{:=}\delta]$$

We will write $\to_\mu$ for the one-step reduction relation induced by the three notions of reduction as above. Similarly, we will respectively write $\twoheadrightarrow_\mu$ and $=_\mu$ for the reflexive, transitive closure, and the reflexive, transitive, symmetric closure of the one-step reduction relation.

The type system of the $\lambda\mu$-calculus is defined by means of a classical sequent calculus. The sequents are either of the form $M \;:\; \Gamma \vdash \Delta$ or of the form $M \;:\; \Gamma \vdash A, \Delta$. In such sequents, M is a $\lambda\mu$-term; the antecedent Γ is a set of second-order formulas indexed by λ-variables; the succedent Δ is a set of second-order formulas indexed by μ-variables; A is a non-indexed second-order formula. The typing rules are the following:

Logical rules

$$x \;:\; A^x \vdash A$$

$$\frac{M \;:\; \Gamma,\, A^x \vdash B,\, \Delta}{\lambda x.\, M \;:\; \Gamma \vdash A \to B,\, \Delta} \qquad \frac{M \;:\; \Gamma \vdash A \to B,\, \Delta \qquad N \;:\; \Pi \vdash A,\, \Sigma}{M\, N \;:\; \Gamma,\, \Pi \vdash B,\, \Delta,\, \Sigma}$$

$$\frac{M \;:\; \Gamma \vdash A[x{:=}y],\, \Delta}{M \;:\; \Gamma \vdash \forall x.A,\, \Delta} \qquad \frac{M \;:\; \Gamma \vdash \forall x.A,\, \Delta}{M \;:\; \Gamma \vdash A[x{:=}t],\, \Delta}$$

$$\frac{M \;:\; \Gamma \vdash A[X{:=}Y],\, \Delta}{M \;:\; \Gamma \vdash \forall X.A,\, \Delta} \qquad \frac{M \;:\; \Gamma \vdash \forall X.A,\, \Delta}{M \;:\; \Gamma \vdash A[X{:=}B],\, \Delta}$$

where the *eigenvariablen* y and Y must obey the usual proviso.

Naming rules

$$\frac{M \;:\; \Gamma \vdash A,\, \Delta}{[\,\alpha\,]\, M \;:\; \Gamma \vdash A^\alpha,\, \Delta} \qquad \frac{M \;:\; \Gamma \vdash A^\alpha,\, \Delta}{\mu\alpha.\, M \;:\; \Gamma \vdash A,\, \Delta}$$

When a judgement of the form $M :\vdash A$ is derivable according to these typing rules, we will write $\vdash_\mu\; M : A$.

If one forgets about the naming rules (and therefore, about μ-abstractions and named terms), the above type system amounts to Krivine's AF_2 [7]. When

dealing with pure λ-calculus or with AF_2, we will allow only for the notion of reduction β. Then, we will respectively write $\rightarrow_\lambda$, $\twoheadrightarrow_\lambda$, and $=_\lambda$ for the relations of one-step reduction, reduction, and conversion induced by β. When dealing with the intuitionistic sequents of AF_2, we will use the standard notation, that is $\Gamma \vdash M : A$ instead of $M \,:\, \Gamma \vdash A$. Finally, when a judgement of the form $\vdash \quad M : A$ is derivable according to the typing rules of AF_2, we will write $\vdash_{\mathrm{AF}_2} M : A$.

3 The CPS-translation

There is a strong connection between classical proofs and control operators like scheme's `call/cc` [6]. Control operators act on the evaluation ordering of programs. Similarly, a feature such as the μ-abstraction may be seen as a construct allowing the evaluation context of a term to be changed by passing over a stack of arguments to some subterms.

Control and evaluation ordering may be modeled in pure λ-calculus by using a technique known as the continuation passing style (CPS, for short) [13]. It is therefore not too surprising that the same technique may be used to give a translation of the $\lambda\mu$-calculus into the λ-calculus.

For the purpose of this CPS-translation, we consider that pure λ-terms are built upon an alphabet of variables made of the sets of λ- and μ-variables of the $\lambda\mu$-calculus plus two distinguished variables, k and m.[2]

Definition 3.1 (CPS-Translation) *The CPS-translation $\underline{M}$ of a $\lambda\mu$-term M is inductively defined as follows:*

(i) $\underline{x} \;=\; \lambda k.\, x\, k$;
(ii) $\underline{\lambda x.\, M} \;=\; \lambda k.\, k\, (\lambda x.\, \underline{M})$;
(iii) $\underline{M\, N} \;=\; \lambda k.\, \underline{M}\, (\lambda m.\, m\, \underline{N}\, k)$;
(iv) $\underline{\mu\delta.\, M} \;=\; \lambda\delta.\, \underline{M}$;
(v) $\underline{[\delta]\, M} \;=\; \lambda k.\, \underline{M}\, \delta\, k$.

This translation, which is based on Plotkin's call-by-name CPS-simulation [13], gives an interesting interpretation of the operations of μ-abstraction and naming. A μ-abstraction $\mu\delta.\, M$ corresponds to a λ-abstraction waiting for some continuation represented by the formal parameter δ. Then, the naming of a subterm corresponds simply to the application of the given subterm to the continuation δ.

It remains to show that our translation is correct with respect to equality. More precisely, we intend to prove the following proposition:

Proposition 3.2 (Translation) *$\underline{M} =_\lambda \underline{N}$ if and only if $M =_\mu N$, for any $\lambda\mu$-terms M and N.* □

[2] Actually, the variables k and m will always occur bound. Nevertheless, for a technical reason that will appear in the sequel, we consider their names as relevant.

We first establish the if part of this proposition. To this end, a couple of lemmas are needed.

Lemma 3.3 *$\lambda k.\, \underline{M}\, k =_\lambda \underline{M}$, for any $\lambda\mu$-term M.*

Proof. The CPS-translation $\underline{M}$ of any $\lambda\mu$-term M is a λ-abstraction. □

Lemma 3.4 *$\underline{M}[x{:=}\underline{N}] =_\lambda \underline{M[x{:=}N]}$, for any $\lambda\mu$-terms M and N.*

Proof. A straightforward induction on the structure of M, using Lemma 3.3 when M is x. □

Lemma 3.5 *$\underline{M}[\delta{:=}\lambda m.\, m\, \underline{N}\, \delta] =_\lambda \underline{M[\delta \Leftarrow N]}$, for any $\lambda\mu$-terms M and N.*

Proof. By induction on the structure of M. We consider only the case $M = [\,\delta\,]\, M'$, the other cases being straightforward.

$$\begin{aligned}
\underline{[\,\delta\,]\, M'}[\delta{:=}\lambda m.\, m\, \underline{N}\, \delta] &= (\lambda k.\, \underline{M'}\, \delta\, k)[\delta{:=}\lambda m.\, m\, \underline{N}\, \delta] \\
&= \lambda k.\, \underline{M'}[\delta{:=}(\lambda m.\, m\, \underline{N}\, \delta)]\, (\lambda m.\, m\, \underline{N}\, \delta)\, k \\
&=_\lambda \lambda k.\, \underline{M'[\delta \Leftarrow N]}\, (\lambda m.\, m\, \underline{N}\, \delta)\, k \quad \text{by induction hypothesis} \\
&=_\lambda \lambda k.\, (\lambda k.\, \underline{M'[\delta \Leftarrow N]}\, (\lambda m.\, m\, \underline{N}\, k))\, \delta\, k \\
&= \lambda k.\, \underline{M'[\delta \Leftarrow N]\, N}\, \delta\, k \\
&= \underline{[\,\delta\,]\, (M'[\delta \Leftarrow N]\, N)} \\
&= \underline{([\,\delta\,]\, M)[\delta \Leftarrow N]}
\end{aligned}$$

□

Thanks to the above lemmas, the proof of the if part of Proposition 3.2 becomes rather easy:

Proof of Proposition 3.2 (if part). The proof is by induction on the derivation of $M =_\mu N$. The inductive steps being straightforward, we will only focus on the base cases. We consider the case of structural reduction step below, and we leave the two other cases to the reader.

$$\begin{aligned}
\underline{(\mu\delta.\, M)\, N} &= \lambda k.\, (\lambda\delta.\, \underline{M})\, (\lambda m.\, m\, \underline{N}\, k) \\
&=_\alpha \lambda\delta.\, (\lambda\delta.\, \underline{M})\, (\lambda m.\, m\, \underline{N}\, \delta) \\
&=_\lambda \lambda\delta.\, \underline{M}[\delta{:=}\lambda m.\, m\, \underline{N}\, \delta] \\
&=_\lambda \lambda\delta.\, \underline{M[\delta \Leftarrow N]} && \text{by Lemma 3.5} \\
&= \underline{\mu\delta.\, M[\delta \Leftarrow N]}
\end{aligned}$$

□

To establish the only-if part of Proposition 3.2 is more intricate. The proof that we will give is adapted from a proof by G. Plotkin in [13]. Let us first explain where the difficulties are.

The proof of the if part of Proposition 3.2 demonstrates that the contraction of a β-redex in a $\lambda\mu$-term is simulated by a sequence of contractions in the CPS-transform. Among these contractions, only one corresponds to the original

contraction in the $\lambda\mu$-term. The other ones, which are related to the management of the continuations, are what G. Plotkin calls *administrative* reduction-steps.

Because of these administrative reductions, we do not have that, when $\underline{M} \twoheadrightarrow_\lambda N$, there is a N' such that $\underline{N'} = N$. To circumvent this problem, we will define a binary relation $M \sim N$ between $\lambda\mu$-terms and λ-terms, with the meaning that N may be obtained from $\underline{M}$ by administrative reductions.

In order to define the relation $\sim$, we must be able to distinguish proper reductions from administrative ones. For this reason, the names of the bound variables k and m introduced in Definition 3.1 will be considered as relevant. For the same reason, the α-conversion step that appears in the simulation of a structural reduction step (see the proof of the if part of Proposition 3.2) will be considered as mandatory.

The relation $\sim$ is defined by the following formal system, where x ranges over λ-variables, δ ranges over μ-variables, and k, m stand for themselves:

$$\text{I.} \quad x \sim \lambda k.\, x\, k$$

$$\text{II.} \quad \frac{M_1 \sim M_2}{\lambda x.\, M_1 \sim \lambda k.\, k\, (\lambda x.\, M_2)}$$

$$\text{III.a.} \quad \frac{M_1 \sim M_2 \quad N_1 \sim N_2}{M_1\, N_1 \sim \lambda k.\, M_2\, (\lambda m.\, m\, N_2\, k)} \qquad \text{III.b.} \quad \frac{M_1 \sim \lambda k.\, M_2 \quad N_1 \sim N_2}{M_1\, N_1 \sim \lambda k.\, M_2[k{:=}\lambda m.\, m\, N_2\, k]}$$

$$\text{IV.} \quad \frac{M_1 \sim M_2 \quad N_1 \sim N_2}{(\lambda x.\, M_1)\, N_1 \sim \lambda k.\, (\lambda x.\, M_2)\, N_2\, k}$$

$$\text{V.} \quad \frac{M_1 \sim M_2}{M_1 \sim \lambda k.\, M_2\, k}$$

$$\text{VI.} \quad \frac{M_1 \sim M_2}{\mu\delta.\, M_1 \sim \lambda\delta.\, M_2}$$

$$\text{VII.a.} \quad \frac{M_1 \sim M_2}{[\,\delta\,]\, M_1 \sim \lambda k.\, M_2\, \delta\, k} \qquad \text{VII.b.} \quad \frac{M_1 \sim \lambda k.\, M_2}{[\,\delta\,]\, M_1 \sim \lambda k.\, M_2[k{:=}\delta]\, k}$$

The advantage of the above definition is that we may now establish the two following key properties:

$$\textit{If } M_1 \sim N_1 \textit{ and } N_1 \to_\lambda N_2 \textit{ then } M_1 \twoheadrightarrow_\mu M_2 \textit{ and } M_2 \sim N_2\textit{, for some } M_2. \quad (1)$$

$$\textit{If } M_1 \sim N \textit{ and } M_2 \sim N \textit{ then } M_1 = M_2. \quad (2)$$

The three next lemmas concern Property 1.

Lemma 3.6 *Let M_1, N_1 be $\lambda\mu$-terms and M_2, N_2 be λ-terms. If $M_1 \sim M_2$ and $N_1 \sim N_2$, then $M_1[x{:=}N_1] \sim M_2[x{:=}N_2]$.*

Proof. A straightforward induction on the derivation of $M_1 \sim M_2$. □

Lemma 3.7 *Let M_1, N_1 be $\lambda\mu$-terms and M_2, N_2 be λ-terms. If $M_1 \sim M_2$ and $N_1 \sim N_2$, then $M_1[\delta \Leftarrow N_1] \sim M_2[\delta{:=}\lambda m.\, m\, N_2\, \delta]$.*

Proof. By induction on the derivation of $M_1 \sim M_2$. The only cases that are not straightforward are those for which $M_1 \sim M_2$ is the conclusion of Rule VII.a or Rule VII.b. These cases are similar to the case $M = [\delta]\, M'$ in the proof of Lemma 3.5. □

Lemma 3.8 (Property 1) *Let M_1 be a $\lambda\mu$-term and N_1, N_2 be λ-terms. If $M_1 \sim N_1$ and $N_1 \to_\lambda N_2$, then there exists a $\lambda\mu$-term M_2 such that $M_1 \twoheadrightarrow_\mu M_2$ and $M_2 \sim N_2$.*

Proof. By induction on the derivation of $M_1 \sim N_1$, using Lemmas 3.6 and 3.7. When $M_1 \sim N_1$ is the conclusion of Rule III.b, a secondary induction is needed. □

Property 2 will be established by induction on the definition of the relation $\sim$. In order to distinguish between different subcases when performing this induction, we must first characterize the form of the λ-terms that belong to the codomain of the relation $\sim$. Consider the following grammar, where x ranges over λ-variables, δ ranges over μ-variables, and k, m stand for themselves:

$$
\begin{array}{lcl}
\mathcal{A} & ::= & \lambda k.\, x\, k \\
\mathcal{B} & ::= & \lambda k.\, k\, (\lambda x.\, \mathcal{H}) \\
\mathcal{C} & ::= & \lambda k.\, x\, \mathcal{K} \;\mid\; \lambda k.\, \mathcal{K}\, (\lambda x.\, \mathcal{H}) \;\mid\; \lambda k.\, \mathcal{H}\, \mathcal{K} \;\mid\; \lambda k.\, (\lambda x.\, \mathcal{H})\, \mathcal{H}\, \mathcal{K} \;\mid\; \lambda k.\, \mathcal{G}'\, \mathcal{K} \\
\mathcal{K} & ::= & \lambda m.\, m\, \mathcal{H}\, k \;\mid\; \lambda m.\, m\, \mathcal{H}\, \mathcal{K} \\
\mathcal{D} & ::= & \lambda k.\, (\lambda x.\, \mathcal{H})\, \mathcal{H}\, k \\
\mathcal{E} & ::= & \lambda k.\, \mathcal{H}\, k \\
\mathcal{F} & ::= & \lambda \delta.\, \mathcal{H} \\
\mathcal{G} & ::= & \lambda k.\, \mathcal{G}'\, k \\
\mathcal{G}' & ::= & x\, \Delta \;\mid\; \Delta\, (\lambda x.\, \mathcal{H}) \;\mid\; \mathcal{H}\, \Delta \;\mid\; (\lambda x.\, \mathcal{H})\, \mathcal{H}\, \Delta \;\mid\; \mathcal{G}'\, \Delta \\
\Delta & ::= & \delta \;\mid\; \lambda m.\, m\, \mathcal{H}\, \Delta \\
\mathcal{H} & ::= & \mathcal{A} \;\mid\; \mathcal{B} \;\mid\; \mathcal{C} \;\mid\; \mathcal{D} \;\mid\; \mathcal{E} \;\mid\; \mathcal{F} \;\mid\; \mathcal{G}
\end{array}
$$

This grammar characterizes the codomain of the relation $\sim$ as follows:

Lemma 3.9 *Let M be a $\lambda\mu$-term and N be a λ-term. If $M \sim N$ then $N \in \mathcal{H}$. Moreover, $N \in \mathcal{A}$ iff the last rule used in the derivation of $M \sim N$ is I; $N \in \mathcal{B}$ iff the last rule is II; $N \in \mathcal{C}$ iff the last rule is III.a or III.b; $N \in \mathcal{D}$ iff the last rule is IV; $N \in \mathcal{E}$ iff the last rule is V; $N \in \mathcal{F}$ iff the last rule is VI; and $N \in \mathcal{G}$ iff the last rule is VII.a or VII.b.*

Proof. The proof is by induction on the derivation of $M \sim N$, the different cases and subcases are numerous but not difficult. □

Thanks to the above lemma, Property 2 may be now established:

Lemma 3.10 (Property 2) *Let M_1, M_2 be a $\lambda\mu$-terms, and N be a λ-term. If $M_1 \sim N$ and $M_2 \sim N$ then $M_1 = M_2$ (where "=" stands for syntactic equivalence modulo α-conversion).*

Proof. By induction on the derivation of $M_1 \sim N$.

By Lemma 3.9, when the last rule of the derivation of $M_1 \sim N$ is respectively I, II, IV, V, or VI, so is the last rule of the derivation of $M_2 \sim N$. Moreover, the form of N determines univocally the forms of the premises. Therefore, in these cases, the induction is straightforward.

If the last rule of the derivation of $M_1 \sim N$ is III.a or III.b, the same reasoning applies except when $N = \lambda k.\, N_1\, (\lambda m.\, m\, N_2\, k)$ because there is a possibility of overlapping. Nevertheless, in this last case, the two possible derivations are the following:

$$\text{III.a.}\ \frac{\begin{array}{cc}\vdots & \vdots\\ M_{11} \sim N_1 & M_{12} \sim N_2\end{array}}{M_{11}\, M_{12} \sim \lambda k.\, N_1\, (\lambda m.\, m\, N_2\, k)} \qquad \text{III.b.}\ \frac{\begin{array}{cc}\vdots & \\ \overline{M_{21} \sim N_1} & \vdots\\ M_{21} \sim \lambda k.\, N_1\, k & M_{22} \sim N_2\end{array}}{M_{21}\, M_{22} \sim \lambda k.\, N_1\, (\lambda m.\, m\, N_2\, k)}\ \text{V.}$$

Therefore, for this case, the induction is also straightforward.

Finally, when the last rule of the derivation is VII.a or VII.b, the only possibility of overlapping is similar to the previous one, and the induction is also straightforward. □

We are now in the position of proving the only-if part of Proposition 3.2:

Proof of Proposition 3.2 (only-if part). Let $\underline{M} =_\lambda \underline{N}$. By the theorem of Church-Rosser, there exists a λ-term O such that $\underline{M} \twoheadrightarrow_\lambda O$ and $\underline{N} \twoheadrightarrow_\lambda O$. As $M \sim \underline{M}$, by Lemma 3.8, there is a $\lambda\mu$-term M' such that $M \twoheadrightarrow_\mu M'$ and $M' \sim O$. Similarly, there is a $\lambda\mu$-term N' such that $N \twoheadrightarrow_\mu N'$ and $N' \sim O$. Then, by Lemma 3.10, $M' = N'$. Therefore, $M =_\mu N$. □

4 Simulation

The proposition established in the previous section is concerned with the correction of the CPS-translation with respect to equality. If one considers the $\lambda\mu$-calculus as a programming language, this criterion of correction is not sufficient any more because it does not say anything about evaluation.

In this section, we intend to answer the following natural question: does the CPS-transform simulate faithfully the evaluation of the $\lambda\mu$-terms. More precisely, is there a mapping Φ such that:

$$\mathrm{eval}_\lambda(\underline{M}) = \Phi(\mathrm{eval}_\mu(M)) \tag{1}$$

In order to give a meaning to this equation, we must first define the functions of evaluation. Traditionally, this is done by giving some abstract machines. In order to be machine independent, we simply define $\mathrm{eval}_\mu(M)$ as the normal form of M, if any. We define $\mathrm{eval}_\lambda(N)$ similarly. These definitions are not ambiguous because both calculi satisfy the Church-Rosser property. They also make sense from an operational point of view, the evaluation functions corresponding to a call-by-name strategy [13].

The two evaluation functions are partial. For this reason, we must also precise the meaning of the equality in (1). We adopt Kleene's complete equality, that is: if one member of the equation is defined, so is the other one and their values are the same. Therefore, in proving an equation similar to (1), we must first establish that the left-hand side is defined if and only if the right-hand side is defined. If Φ is a total function, this amounts to show that a $\lambda\mu$-term is normalizable if and only if its CPS-translation is normalizable. Half of this property may be proven with the tools that we have introduced in the previous section.

Lemma 4.1 *Let M be a $\lambda\mu$-term and N be a λ-term such that $M \sim N$. If N is in λ-normal form then M is in μ-normal form.*

Proof. By induction on the derivation of $M \sim N$, using Lemma 3.9 to distinguish between the different subcases. □

Lemma 4.2 *Let M be a $\lambda\mu$-term. If $\underline{M}$ is λ-normalizable then M is μ-normalizable.*

Proof. Let Z be the λ-normal form of $\underline{M}$. As $M \sim \underline{M}$, by Lemma 3.8, there is a $\lambda\mu$-term Y such that $M \twoheadrightarrow_\mu Y$ and $Y \sim Z$. Then, by Lemma 4.1, Y is the μ-normal form of M. □

To establish that $\underline{M}$ is normalizable whenever M is, we must face two problems. The first one is related to the administrative reductions: in general, when M is in μ-normal form, we do not have that $\underline{M}$ is in λ-normal form. The second one is related to the simulation of the structural reduction: if $M_1 \to_\mu M_2$ by structural reduction, we do not have that $\underline{M_1} \twoheadrightarrow_\lambda \underline{M_2}$; this is due to the β-expansion step that appears in the proof of Lemma 3.5.

The solution to these problems is to modify the definition of the CPS-transform in such a way that some of the administrative redexes disappear:

Definition 4.3 (Modified CPS-Translation) *The modified CPS-translation $\underline{\underline{M}}$ of a $\lambda\mu$-term M is inductively defined as follows:*

(i) $\underline{\underline{x}} = \lambda k.\, x\, k$;

(ii) $\underline{\underline{\lambda x.\, M}} = \lambda k.\, k\, (\lambda x.\, \underline{\underline{M}})$;

(iii) $\underline{\underline{M\, N}} = \lambda k.\, (\underline{\underline{M}} : \lambda m.\, m\, \underline{\underline{N}}\, k)$;

(iv) $\underline{\underline{\mu\delta.\, M}} = \lambda\delta.\, \underline{\underline{M}}$;

(v) $\underline{\underline{[\delta]\, M}} = \lambda k.\, \overline{[\delta]\, M}\, k$;

where:

(vi) $x : K \;=\; x\, K$;
(vii) $(\lambda x.\, M) : K \;=\; K\, (\lambda x.\, \underline{\underline{M}})$;
(viii) $(M\, N) : K \;=\; M : \lambda m.\, m\, \underline{\underline{N}}\, K$;
(ix) $(\mu\delta.\, M) : K \;=\; (\lambda\delta.\, \underline{\underline{M}})\, K$;
(x) $([\,\delta\,]\, M) : K \;=\; \overline{[\,\delta\,]\, M} : K$;
(xi) $\overline{[\,\delta\,]\, x} \;=\; x\, \delta$;
(xii) $\overline{[\,\delta\,]\, (\lambda x.\, M)} \;=\; \delta\, (\lambda x.\, \underline{\underline{M}})$;
(xiii) $\overline{[\,\delta\,]\, (M\, N)} \;=\; M : (\lambda m.\, m\, \underline{\underline{N}}\, \delta)$;
(xiv) $\overline{[\,\delta\,]\, (\mu\gamma.\, M)} \;=\; (\lambda\gamma.\, \underline{\underline{M}})\, \delta$;
(xv) $\overline{[\,\delta\,]\, ([\gamma]\, M)} \;=\; \overline{[\,\gamma\,]\, M}\, \delta$.

This new definition is compatible with the previous one in the following sense:

Lemma 4.4 *$\underline{M} \twoheadrightarrow_\lambda \underline{\underline{M}}$, for any $\lambda\mu$-term M.*

Proof. By induction on the structure of M, using auxiliary inductions when M is an application or a named term. □

As expected, the modified CPS-transform maps normal forms to normal forms:

Lemma 4.5 *Let M be a $\lambda\mu$-term in μ-normal form. Then $\underline{\underline{M}}$ is in λ-normal form.*

Proof. By induction on the structure of M. □

These two lemmas allow us to show that if M is normalizable, so is $\underline{M}$:

Lemma 4.6 *Let M be a $\lambda\mu$-term. If M is μ-normalizable then $\underline{M}$ is λ-normalizable.*

Proof. Let Z be the μ-normal form of M. By Proposition 3.2, $\underline{M} =_\lambda \underline{Z}$. On the other hand, by Lemma 4.4, $\underline{Z} \twoheadrightarrow_\lambda \underline{\underline{Z}}$, and this last term is in β-normal form by Lemma 4.5. Finally, by Church-Rosser, $\underline{M} \twoheadrightarrow_\lambda \underline{\underline{Z}}$. □

Lemmas 4.2, 4.4, 4.5, and 4.6, allow us to prove Equation (1) where the mapping Φ is nothing but the modified CPS-transform. Therefore, we have the following:

Proposition 4.7 (Simulation) $\mathrm{eval}_\lambda(\underline{M}) = \underline{\underline{\mathrm{eval}_\mu(M)}}$, *for any $\lambda\mu$-term M.* □

We end this section by a remark. Traditionally, the evaluation of a program is simulated by the evaluation of its CPS-translation applied to the empty continuation $\lambda x.\, x$. This make sense because a program is defined to be a closed term of atomic type. Now, the only closed normal form of atomic types are basic constants and the CPS-transform of a basic constant a is defined to be $\lambda k.\, k\, a$. Therefore, for any program P, we have that $\mathrm{eval}\,(P) = \mathrm{eval}\,(\underline{P}\,(\lambda x.\, x))$.

We did not follow this traditional approach for two reasons. The first one is that in the $\lambda\mu$-calculus, as in AF_2, the basic data-structures are not of atomic

type. For instance, the type of natural numbers is represented by the following second order formula that asserts that n is a natural number:

$$\forall X.(X(0) \to \forall y.(X(y) \to X(sy)) \to X(n)).$$

The second reason, as we will see in the next section, is that $\underline{M}\,(\lambda x.\, x)$ is not always well-typed.

5 Logical Interpretation

Up to now, we have worked in the framework of the untyped $\lambda\mu$-calculus. In this section, we intend to answer questions such as the following ones: does the CPS-translation make sense in a typed framework? is it typable? if yes, does it induce some interesting translation at the type level? These different questions are summarized by the following one: is there a translation acting on the types such that this translation and the CPS-translation would commute with the typing relations of $\lambda\mu$ and AF_2?

The answer is positive. The translation at the type level corresponds to a logical interpretation of classical logic into intuitionistic one akin to Kolmogorov's negative translation.

Definition 5.1 (Negative Translation) *The negative translation A^{k} of a first-order formula A is inductively defined as follows:*

(i) $A^{\mathrm{k}} = \neg\neg A$, *(for A an atomic formula);*
(ii) $(A \to B)^{\mathrm{k}} = \neg\neg(A^{\mathrm{k}} \to B^{\mathrm{k}})$;
(iii) $(\forall x.A)^{\mathrm{k}} = \neg\neg\forall x.A^{\mathrm{k}}$;
(iv) $(\forall X.A)^{\mathrm{k}} = \neg\neg\forall X.A^{\mathrm{k}}$;

where $\neg A = A \to \mathbf{f}$, and $\mathbf{f} = (\bot \to \bot) \to \bot$.

The proposition that we are going to establish in this section is the following:

Proposition 5.2 (Logical Interpretation) *If $\vdash_{\mu} M : A$ then $\vdash_{\mathrm{AF}_2} \underline{M} : A^{\mathrm{k}}$, for any $\lambda\mu$-term M and any second-order formula A.* □

In trying to prove the above proposition, we will encounter a problem related to the introduction of the quantifiers. Indeed, whenever $\vdash_{\mathrm{AF}_2} \underline{M} : A^{\mathrm{k}}$, we may conclude that $\vdash_{\mathrm{AF}_2} \underline{M} : \forall x.A^{\mathrm{k}}$. But this is not sufficient because one does not have that $(\forall x.A)^{\mathrm{k}} = \forall x.A^{\mathrm{k}}$.

A solution to this problem is to extend Definition 3.1 by defining the CPS-translation of the typed $\lambda\mu$-terms by induction on the derivations of the typing judgments. The only typing rules that are not reflected by the syntax of the $\lambda\mu$-terms are the ones related to the quantifiers. Therefore, we extend Definition 3.1 by adding the two following clauses:

(vi) $\underline{M} = \lambda k.\, k\, \underline{M}$, *if M is obtained by first- or second-order $\forall$-introduction;*

(vii) $\underline{M} = \lambda k.\, \underline{M}\, (\lambda m.\, m\, k)$, *if M is obtained by first- or second-order $\forall$-elimination.*

Then Propositions 3.2 and 4.7 must be restated in terms of proof reduction. This gives rise to the following additional reduction steps (we give them for the first-order case, the second-order case being completely similar):

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ M \ :\ \Gamma \vdash A,\, \Delta\end{array}}{M \ :\ \Gamma \vdash \forall x.A,\, \Delta}}{M \ :\ \Gamma \vdash A[x{:=}t],\, \Delta} \quad \text{reduces to} \quad \dfrac{\begin{array}{c}\mathbf{\Pi}[x{:=}t]\\ \vdots\\ M \ :\ \Gamma \vdash A[x{:=}t],\, \Delta\end{array}}{}$$

This logical reduction step is simulated by the reduction of the following λ-term:

$$\lambda k.\, (\lambda k.\, k\, \underline{M})\, (\lambda m.\, m\, k).$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ M_i \ :\ \Gamma_i \vdash \forall x.A,\, \Delta_i\end{array}}{[\alpha]\, M_i \ :\ \Gamma_i \vdash \forall x.A^{\alpha},\, \Delta_i}}{\begin{array}{c}\vdots\\ M \ :\ \Gamma \vdash \forall x.A^{\alpha},\, \Delta\end{array}}}{\mu\alpha.\, M \ :\ \Gamma \vdash \forall x.A,\, \Delta}}{\mu\alpha.\, M \ :\ \Gamma \vdash A[x{:=}t],\, \Delta} \quad \text{reduces to} \quad \dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ M_i \ :\ \Gamma_i \vdash \forall x.A,\, \Delta_i\end{array}}{M_i \ :\ \Gamma_i \vdash A[x{:=}t],\, \Delta_i}}{[\alpha]\, M_i \ :\ \Gamma_i \vdash A[x{:=}t]^{\alpha},\, \Delta_i}}{\begin{array}{c}\vdots\\ M \ :\ \Gamma \vdash A[x{:=}t]^{\alpha},\, \Delta\end{array}}}{\mu\alpha.\, M \ :\ \Gamma \vdash A[x{:=}t],\, \Delta}$$

This structural reduction step is simulated by the reduction of the following λ-term:

$$\lambda k.\, (\lambda \alpha.\, \underline{M})\, (\lambda m.\, m\, k).$$

In order to prove Proposition 5.2, we must be able to represent classical sequents by intuitionistic ones. A possible solution to this problem is to negate each formula of the succedent and to add this sequence of negated formulas to the antecedent. In our case, we may not apply this idea roughly because we do not want to deal with triple negations. Therefore, given a second-order formula A, we define $A^{\mathrm{k}-}$ to be the unique formula such that $A^{\mathrm{k}} = \neg A^{\mathrm{k}-}$.

Proof of Proposition 5.2 We interpret any sequent

$$M \ :\ \Gamma \vdash A,\, \Delta \tag{1}$$

of the $\lambda\mu$-calculus by the following intuitionistic sequent of AF_2:

$$\Gamma^{\mathrm{k}},\, \Delta^{\mathrm{k}-} \vdash \underline{M} : A^{\mathrm{k}}, \tag{2}$$

and we show that if (1) is derivable so is (2).

Given a second-order formula A, we define A^* to be the formula such that $A^{\mathrm{k}} = \neg\neg A^*$. Then, we have that $A^{\mathrm{k}-} = \neg A^*$. We will use these notations in the sequel of the proof, which is done by induction on the derivation of (1). We only handle some interesting cases, leaving the other ones to the reader.

Elimination of second-order quantification:

$$\mathbf{\Pi}\left\{\begin{array}{c}\dfrac{\dfrac{m : \forall X.A^{\mathrm{k}} \vdash m : \forall X.A^{\mathrm{k}}}{m : \forall X.A^{\mathrm{k}} \vdash m : \neg\neg A^*[X{:=}B^*]} \quad k : \neg A^*[X{:=}B^*] \vdash k : \neg A^*[X{:=}B^*]}{m : \forall X.A^{\mathrm{k}},\ k : \neg A^*[X{:=}B^*] \vdash m\,k : \mathbf{f}} \\ \hline k : \neg A^*[X{:=}B^*] \vdash \lambda m.\, m\,k : \neg\forall X.A^{\mathrm{k}}\end{array}\right.$$

$$\dfrac{\begin{array}{ll}\textit{induction hypothesis} & \mathbf{\Pi} \\ \vdots & \vdots \\ \Gamma^{\mathrm{k}},\ \Delta^{\mathrm{k}-} \vdash \underline{M} : \neg\neg\forall X.A^{\mathrm{k}} & k : \neg A^*[X{:=}B^*] \vdash \lambda m.\, m\,k : \neg\forall X.A^{\mathrm{k}}\end{array}}{\dfrac{\Gamma^{\mathrm{k}},\ \Delta^{\mathrm{k}-},\ k : \neg A^*[X{:=}B^*] \vdash \underline{M}\,(\lambda m.\, m\,k) : \mathbf{f}}{\Gamma^{\mathrm{k}},\ \Delta^{\mathrm{k}-} \vdash \lambda k.\, \underline{M}\,(\lambda m.\, m\,k) : (A[X{:=}B])^{\mathrm{k}}}}$$

μ-Abstraction:

$$\dfrac{\begin{array}{c}\textit{induction hypothesis} \\ \vdots \\ \Gamma^{\mathrm{k}},\ \Delta^{\mathrm{k}-},\ \delta : \neg A^* \vdash \underline{M} : \mathbf{f}\end{array}}{\Gamma^{\mathrm{k}},\ \Delta^{\mathrm{k}-} \vdash \lambda\delta.\, \underline{M} : \neg\neg A^*}$$

Naming:

$$\dfrac{\dfrac{\begin{array}{c}\textit{induction hypothesis} \\ \vdots \\ \Gamma^{\mathrm{k}},\ \Delta^{\mathrm{k}-} \vdash \underline{M} : \neg\neg A^*\end{array} \qquad \delta : \neg A^* \vdash \delta : \neg A^*}{\Gamma^{\mathrm{k}},\ \Delta^{\mathrm{k}-},\ \delta : \neg A^* \vdash \underline{M}\,\delta : \mathbf{f}} \qquad k : \bot \to \bot \vdash k : \bot \to \bot}{\dfrac{\Gamma^{\mathrm{k}},\ \Delta^{\mathrm{k}-},\ \delta : \neg A^*,\ k : \bot \to \bot \vdash \underline{M}\,\delta\,k : \bot}{\Gamma^{\mathrm{k}},\ \Delta^{\mathrm{k}-},\ \delta : \neg A^* \vdash \lambda k.\, \underline{M}\,\delta\,k : \mathbf{f}}}$$

□

The reason why we did define $\neg A$ as $A \to \mathbf{f}$ appears in the last case of the above proof. We could get a simpler negative translation by simplifying, in the definition of the CPS-translation, Clauses (i) and (v) as follows:

(i) $\underline{x} = x$;

(v) $\underline{[\delta]\, M} = \underline{M}\, \delta$.

With this definition, however, Lemma 3.3 does not hold any more and therefore some sort of η-reduction would be needed.[3]

The well-typed terms of AF_2 are normalizable [7]. Therefore, we get the following proposition as a corollary.

Corollary 5.3 (Normalization) *Any well-typed $\lambda\mu$-term is normalizable.* □

6 Conclusions

We have presented a CPS-translation of the $\lambda\mu$-calculus into the λ-calculus that satisfies *translation* and *simulation* properties similar to the ones introduced in [13]. As we pointed out in the introduction, this CPS-translation is general in the sense that it works for any (untyped) $\lambda\mu$-term. Moreover, it maps typed $\lambda\mu$-terms to typed λ-terms and this allows us to get the normalization of the typed $\lambda\mu$-calculus for free.

In [13], G. Plotkin also establishes indifference results with respect to call-by-name and call-by-value strategies. We did not make such an analysis in this paper. Nevertheless, the reader familiar with [13], may check that all the β-redexes that are contracted during the simulation of the evaluation of a $\lambda\mu$-term are in fact β_V-redexes. Therefore our work gives also an interpreter-independent operational semantics to the $\lambda\mu$-calculus. The interest of this result is not only theoretic. Indeed, the continuation passing style is a technique that is actually used in compiling [1].

Finally, we want to stress that the notion of continuation is not only syntactic and that it has been widely used for semantic purposes. While the work that we have presented in this paper remains merely syntactic, it can be the starting point of some semantic investigations of classical proofs. Indeed, our translation allows $\lambda\mu$-terms to be interpreted in λ-algebras.

References

1. A. W. Appel. *Compiling with continuations.* Cambridge University Press, 1992.
2. F. Barbanera and S. Berardi. Continuations and simple types: a strong normalization result. In *Proceedings of the ACM SIGPLAN Workshop on Continuations.* Report STAN-CS-92-1426, Stanford University, 1992.
3. F. Barbanera and S. Berardi. Extracting constructive content from classical logic via control-like reductions. In M. Bezem and J.F. Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 45–59. Lecture Notes in Computer Science, 664, Springer Verlag, 1993.

[3] This problem seems to have been overlooked in [13] where Theorem 6, as it is stated, fails

4. H.P. Barendregt. *The lambda calculus, its syntax and semantics*. North-Holland, revised edition, 1984.
5. J.-Y. Girard. A new constructive logic: Classical logic. *Mathematical Structures in Computer Science*, 1:255–296, 1991.
6. T. G. Griffin. A formulae-as-types notion of control. In *Conference record of the seventeenth annual ACM symposium on Principles of Programming Languages*, pages 47–58, 1990.
7. J.-L. Krivine. *Lambda-calcul, types et modèles*. Masson, 1990.
8. C. R. Murthy. An evaluation semantics for classical proofs. In *Proceedings of the sixth annual IEEE symposium on logic in computer science*, pages 96–107, 1991.
9. C. R. Murthy. A computational analysis of Girard's translation and LC. In *Proceedings of the seventh annual IEEE symposium on logic in computer science*, pages 90–101, 1992.
10. M. Parigot. $\lambda\mu$-Calculus: an algorithmic interpretation of classical natural deduction. In A. Voronkov, editor, *Proceedings of the International Conference on Logic Programming and Automated Reasoning*, pages 190–201. Lecture Notes in Artificial Intelligence, 624, Springer Verlag, 1992.
11. M. Parigot. Classical proofs as programs. In G. Gottlod, A. Leitsch, and D. Mundici, editors, *Proceedings of the third Kurt Gödel colloquium – KGC'93*, pages 263–276. Lecture Notes in Computer Science, 713, Springer Verlag, 1993.
12. M. Parigot. Strong normalization for second order classical natural deduction. In *Proceedings of the eighth annual IEEE symposium on logic in computer science*, pages 39–46, 1993.
13. G. D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theretical Computer Science*, 1:125–159, 1975.