Abstract. We present a translation of Parigot's $\lambda\mu$-calculus [10] into the usual $\lambda$-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 $\lambda$-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 $\mu$-abstraction is interpreted as a $\lambda$-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 Actually, M. Parigot proves more, namely strong normalization.
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 $\lambda$-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 $\lambda$-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 $\lambda$-calculus (addition of $\mu$-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 $\lambda$-variables, and the set $\mu$-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 $\lambda$-variables, and $\delta$ ranges over $\mu$-variables. A $\lambda\mu$-term of the form $\mu \delta. T$ is called a $\mu$-abstraction, and a $\lambda\mu$-term of the form $[\delta] T$ is called a named term. The operator $\mu$ is a binding operator as is $\lambda$. Therefore, the free occurrences of a $\mu$-variable $\delta$ 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 $\mu$-variables as well as for $\lambda$-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 $\beta$:

$$ (\lambda x. M) N \rightarrow M[x:= N] $$

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 $A \rightarrow B$, only the subterms named by $\alpha$ are really of type $A \rightarrow B$ (see the typing rules hereafter); hence, when such a $\mu$-abstraction is applied to an argument, this argument must be passed over to the subterms named by $\alpha$. This intuition is formalized as follows:

$$ (\mu \delta. M) N \rightarrow M[\delta \leftarrow N], $$

where the structural substitution is inductively defined as follows: