On the Relation between the $\lambda\mu$-Calculus and the Syntactic Theory of Sequential Control

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 construct a translation of first order $\lambda\mu$-calculus [15] into a subtheory of Felleisen's λ_c-calculus [5, 6]. This translation preserves typing and reduction. Then, by constructing the inverse translation, we show that the two calculi are actually isomorphic.

1 Introduction

For a long time it has been widely thought that a classical proof, as opposed to an intuitionistic one, did not carry any computational content (see, for instance, [8, App. B, §B.1] and [12, p. 67, Proposition 8.3]). In 1990, however, T. Griffin opened a new research area by introducing a classical formulae-as-types notion of control based on Felleisen's $\mathcal{C}$ operator [9]. Since then, various authors have defined different systems that enlighten the constructive content of classical logic [1, 2, 7, 13, 14, 15, 16, 17].

Despite its originality, Griffin's work has been criticized by some logicians. In [15], for instance, M. Parigot writes that *the system he* (Griffin) *obtains is not satisfactory from the logical point of view: the reduction is in fact a reduction strategy and the type assigned to $\mathcal{C}$ doesn't fit in general the reduction rule for $\mathcal{C}$.* Such criticisms are based on a misunderstanding of Griffin's motivations. His goal was not to define a new calculus but to type an existing one, namely Felleisen's syntactic theory of sequential control (λC, for short).

The possible defects of Griffin's proposal are only due to the fact that the computation rule of a sequential control operator such as $\mathcal{C}$ is inherently context sensitive. In order to push out this context sensitiveness as far as possible, M. Felleisen defines a large part of his calculus by means of usual notions of reduction, i.e., notions of reduction that are compatible with the term formation rules. Therefore, if one consider only the subtheory defined by these congruent notions of reduction, Griffin's typing amounts to a system that satisfies interesting properties such as confluence and subject reduction.

Moreover, from a computational point of view, Griffin-Felleisen system is at least as powerful as first order $\lambda\mu$-calculus since the latter is isomorphic to a subtheory of the former. This is the central result of this paper that we establish by constructing a translation of the $\lambda\mu$-calculus into the λ_c-calculus. This result

In: F. Pfenning (Ed.), *Proceedings of the 5th International Conference on Logic Programming and Automated Reasoning (LPAR'94),*
Lecture Notes in Computer Science, 822, Springer-Verlag, (1994), pp. 31-43.

also demonstrates that Griffin's and Parigot's approaches are not as different as one could think at first sight.

The remainder of this paper is organized as follows. The two next sections introduce briefly Felleisen's λ_c-calculus and Parigot's $\lambda\mu$-calculus ($\lambda\mu$, for short). In Section 4, we define the translation of $\lambda\mu$ into λC, and we study its properties. In Section 5, we construct the inverse translation. Finally, we present our conclusions in Section 6.

All through the paper, we assume that the reader has an elementary knowledge of λ-calculus and natural deduction. We also adopt Barendregt's variable convention, which protects one from clashes between free and bound variables. This background material may be found in standard books such as [3, 8, 10, 18, 19].

2 Felleisen's $\boldsymbol{\lambda_c}$-Calculus

Felleisen's calculus is a call-by-value λ-calculus including a control operator $\mathcal{C}$ akin to Scheme `call/cc` [5, 6]. The core syntax of the language is given by the following grammar:

$$T \ ::= \ x \mid (\lambda x.\, T) \mid (T\, T) \mid (\mathcal{C}\, T),$$

where a λ_c-term of the form $\mathcal{C}\, T$ is called a $\mathcal{C}$*-application*.

The operational semantics of the language may be easily defined by rewriting rules expressed in terms of *applicative contexts*. Let a *value* be defined to be a variable or a λ-abstraction, and let V range over values. Applicative contexts are defined by the following grammar:

$$\mathbf{C} \ ::= \ [\,] \mid V\, \mathbf{C} \mid \mathbf{C}\, T.$$

As usual, [ ] represents a hole, and the expression $\mathbf{C}[M]$ denotes the λ_c-term obtained by putting the λ_c-term M into the hole of the applicative context $\mathbf{C}$.

The rewriting rules defining the operational semantics of the language are the following:

$$\mathbf{C}[(\lambda x.\, M)\, V] \to \mathbf{C}[M[V/x]], \qquad \text{(C1)}$$

$$\mathbf{C}[\mathcal{C}\, M] \to M\, (\lambda x.\, \mathcal{A}\, (\mathbf{C}[x])). \qquad \text{(C2)}$$

In Rule C1, V stands for a value, and $M[V/x]$ denotes the usual capture-avoiding substitution. In Rule C2, $\mathcal{A}$ is the *abort* operator that can be defined as

$$\mathcal{A}\, M \ \overset{\text{def}}{=} \ \mathcal{C}\, (\lambda x.\, M),$$

where x is a dummy variable that does not occur free in M.

It is easy to show that any closed λ_c-term is either a value or can be written in a unique way as $\mathbf{C}[M]$, where M is either a β_v-redex or a $\mathcal{C}$-application. Therefore, Rules C1 and C2 defines a totally deterministic strategy of evaluation, and the applicative context $\mathbf{C}$ that is uniquely determined by each rewriting step may be interpreted as being the *current continuation.* This interpretation allows one to explain intuitively the behavior of the operator $\mathcal{C}$. When evaluated within an applicative context $\mathbf{C}$, a $\mathcal{C}$-application gives complete control to its argument by applying it to a procedural abstraction of the current continuation $\mathbf{C}$. In the sequel, if this procedural abstraction is invoked with a value V, the new current applicative context is abandoned, and control is given to the term $\mathbf{C}[V]$.

While Rules C1 and C2 are useful to define the operational semantics of the language, there are not convenient to reason about programs. For this reason, Felleisen has developed a calculus in which a large part of the operational semantics of his language is expressed by means of proper notions of reduction, that is notions of reduction that are compatible with the term formation rules. These notions of reduction are the following:

$$(\lambda x.\, M)\, V \to M[V/x] \qquad (\beta_v)$$

$$(\mathcal{C}\, M)\, N \to \mathcal{C}\, (\lambda k.\, M\, (\lambda f.\, \mathcal{A}\, (k\, (f\, N)))) \qquad (\mathcal{C}_{\mathrm{L}})$$

$$V\, (\mathcal{C}\, M) \to \mathcal{C}\, (\lambda k.\, M\, (\lambda f.\, \mathcal{A}\, (k\, (V\, f)))) \qquad (\mathcal{C}_{\mathrm{R}})$$

$$\mathcal{C}\, M \to \mathcal{C}\, (\lambda k.\, M\, (\lambda f.\, \mathcal{A}\, (k\, f))) \qquad (\mathcal{C}_{\mathrm{top}})$$

$$\mathcal{C}\, (\lambda k.\, \mathcal{C}\, M) \to \mathcal{C}\, (\lambda k.\, M\, (\lambda f.\, \mathcal{A}\, f)) \qquad (\mathcal{C}_{\mathrm{idem}})$$

The resulting calculus satisfies the Church-Rosser property and a form of the standardization theorem [6]. It also captures a large part of the original operational semantics. In order to get equivalence, only one additional computation rule is needed:

$$\mathcal{C}\, M \rhd M\, (\lambda f.\, \mathcal{A}\, f) \qquad (\mathcal{C}_{\mathrm{T}})$$

Moreover, the application of this computation rule, which is not a proper notion of reduction, may be delayed.

In [9], T. Griffin analyses the rewriting rule C2, and proposes to give the operator $\mathcal{C}$ the type $\neg\neg A \to A$. As stressed by M. Parigot, this type assignment is not preserved by evaluation[1]. The only type violation, however, is due to the computation rule $\mathcal{C}_{\mathrm{T}}$. Therefore, if one considers only the proper notions of reduction, Griffin's type assignment satisfies the subject reduction property.

For the purpose of this paper, we will slightly modify Felleisen's calculus. First of all, Parigot's $\lambda\mu$-calculus is not a call-by-value calculus. Hence, in order to establish our equivalence result, we must abandon Felleisen's call-by-value strategy. In other words, we must replace the notion of reduction β_v by the usual notion of reduction β. By doing this, one destroys the confluence of the calculus because a term of the form

$$(\lambda x.\, M)\, (\mathcal{C}\, N)$$

[1] Griffin, of course, mentions the problem, and solves it by encapsulating each program M into the expression $\mathcal{C}\, (\lambda k.\, k\, M)$.

may now be reduced in two different ways. Therefore, in order to conserve the Church-Rosser property, we will drop the notion of reduction $\mathcal{C}_{\mathrm{R}}$. We will also drop the notion of reduction $\mathcal{C}_{\mathrm{idem}}$, simply because it is not needed.

Another modification concerns the presence of the abort operators in Felleisen's reduction rules. As far as control is concerned, these abort operators are useful. However, from a proof-theoretic point of view, there are irrelevant. In general, the type scheme assigned to the abort operator is $\bot \to A$. In Felleisen's rules, however, the abort operators must all be assigned the trivial type $\bot \to \bot$. Take for instance the notion of reduction $\mathcal{C}_L$. It corresponds to the following proof-reduction step:

$$\dfrac{\dfrac{\neg\neg(A \to B)}{A \to B} \qquad A}{B} \quad \to \quad \dfrac{\dfrac{\dfrac{\neg\neg(A \to B) \qquad \dfrac{\dfrac{\dfrac{[\neg B] \qquad \dfrac{[A \to B] \quad A}{B}}{\bot}}{\bot}}{\neg(A \to B)}}{\bot}}{\neg\neg B}}{B}$$

Clearly, the inference $\frac{\bot}{\bot}$ is useless.

To summarize, the calculus that we will consider is the one induced by the modified notions of reduction that follows:

$$(\lambda x.\, M)\, N \to M[N/x] \tag{β}$$

$$(\mathcal{C}\, M)\, N \to \mathcal{C}\, (\lambda k.\, M\, (\lambda f.\, k\, (f\, N))) \tag{$\mathcal{C}_{\mathrm{L}}$}$$

$$\mathcal{C}\, M \to \mathcal{C}\, (\lambda k.\, M\, (\lambda f.\, k\, f)) \tag{$\mathcal{C}_{\mathrm{top}}$}$$

From now on, when we will speak of Felleisen's calculus (λC), we will mean the subtheory resulting from the three modified notions of reduction β, $\mathcal{C}_{\mathrm{L}}$, and $\mathcal{C}_{\mathrm{top}}$. This calculus satisfies the Church-Rosser property; this can be established by replaying Felleisen's proof because it is based on the Hindley-Rossen lemma. The symbol $\to_c$ will denote the one-step reduction relation of λC; and the symbols $\twoheadrightarrow_c$ and $=_c$ will stand, respectively, for the reflexive, transitive closure and the reflexive, transitive, symmetric closure of the one-step reduction. Finally, when a λ_c-term M is typable with type A according to Griffin's system, we will write $\vdash_C\ M : A$.

3 Parigot's $\boldsymbol{\lambda\mu}$-Calculus

Parigot's $\lambda\mu$-calculus is a classical extension of Krivine's AF_2 [11]. It is therefore a second-order system. In this paper, we will focus on first-order $\lambda\mu$-calculus.

This restriction affects neither the syntax of the language, nor the reduction rules. Simply, it allows fewer terms to be typable.

$\lambda\mu$-Terms are built from two distinct alphabets of variables: the set of λ-variables, and the set of so-called μ-variables. The core syntax of the language is given by the following grammar:

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

where x ranges over λ-variables, and α ranges over μ-variables. A $\lambda\mu$-term of the form $\mu\alpha.\, T$ is called a μ-abstraction, and a $\lambda\mu$-term of the form $[\alpha]\, T$ is called a named term. As λ, μ is binding operator: the free occurrences of a μ-variable α in T become bound in $\mu\alpha.\, T$.

The typing rules are given by means of a classical sequent calculus. The sequents are either of the form $\Gamma \vdash \Delta$ or of the form $\Gamma \vdash A, \Delta$, where the antecedent Γ is a set of formulas indexed by λ-variables, the succedent Δ is a set of formulas indexed by μ-variables, and A is a distinguished formula. The typing rules are the following:

Logical rules

$$x \,:\, A^x \vdash A$$

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

Naming rules

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

In addition to the usual notion of reduction β, there is a *structural* notion of reduction, which is related to μ-abstraction. The intuition behind the operations of naming and μ-abstraction may be 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 $A \to B$; therefore, when such a term is applied to an argument, this argument must be passed over to the subterms named by α. The structural reduction rule formalizes this intuition:

$$(\mu\alpha.\, M)\, N \to \mu\alpha.\, M[N/\!*\, \alpha],$$

where the structural substitution is inductively defined as follows:

(i) $x[N/\!*\, \alpha] \ = \ x$;

(ii) $(\lambda x.\, M)[N/\!*\, \alpha] \ = \ \lambda x.\, M[N/\!*\, \alpha]$;

(iii) $(M\, O)[N/\!*\, \alpha] \ = \ M[N/\!*\, \alpha]\, O[N/\!*\, \alpha]$;

(iv) $(\mu\beta.\, M)[N/\!*\, \alpha] \ = \ \mu\beta.\, M[N/\!*\, \alpha]$;

(v) $([\alpha]\, M)[N/\!*\, \alpha] \ = \ [\alpha]\, (M[N/\!*\, \alpha]\, N)$;

(vi) $([\beta]\, M)[N/\!*\,\alpha] \;=\; [\beta]\, M[N/\!*\,\alpha]$ if $\alpha \neq \beta$.

There are some other notions of reduction, among which *renaming*:

$$[\alpha]\,(\mu\beta.\, M) \to M[\alpha/\beta]$$

According to M. Parigot himself, such notions of reduction *are essentially trivial from a computational viewpoint* [16]. Moreover, like the η-reduction steps in λ-calculus, they can be postponed with respect to the structural and the β-reduction steps. We will not allow for them.

In [15], M. Parigot proposes the two following rules to handle the negation:

$$\frac{M \;:\; \Gamma,\, A^x \vdash \Delta}{\lambda x.\, \mu\alpha.\, M \;:\; \Gamma \vdash \neg A,\, \Delta} \qquad \frac{M \;:\; \Gamma \vdash \neg A,\, \Delta \qquad N \;:\; \Pi \vdash A,\, \Sigma}{[\beta]\,(M\, N) \;:\; \Gamma,\, \Pi \vdash \Delta,\, \Sigma},$$

where the μ-variable α is fresh. These rules, which are based on an explicit treatment of *falsum* ($\perp$), present some peculiarities.

On the one hand, the occurrence of the μ-variable β introduced by the elimination rule will desperately remain free. Therefore judgment such as the following are derivable:

$$\lambda y.\, \mu\alpha.\, [\phi]\,(y\,(\lambda x.\, \mu\delta.\, [\alpha]\, x)) \;:\vdash \neg\neg A \to A$$

This is quite surprising: while the above $\lambda\mu$-term stands for a completed proof, it contains a free μ-variable (something like a useless hypothesis that has not been discarded).

On the other hand, unlike the other logical rules, the rules for the negation do not correspond to the intuitionistic ones.

For these reasons, we will substitute the following alternative rules for the original ones:

$$\frac{M \;:\; \Gamma,\, A^x \vdash \Delta}{\lambda x.\, M \;:\; \Gamma \vdash \neg A,\, \Delta} \qquad \frac{M \;:\; \Gamma \vdash \neg A,\, \Delta}{M\, x \;:\; \Gamma,\, A^x \vdash \Delta}.$$

These rules are based on an implicit treatment of *falsum* that amounts to identify any sequent of the form $\Gamma \vdash \Delta$ with the sequent $\Gamma \vdash \perp,\, \Delta$.

From now on, when we will speak of the $\lambda\mu$-calculus ($\lambda\mu$), we will mean first-order $\lambda\mu$-calculus, with the structural notion of reduction and the usual notion of reduction β, and with the alternative rules for negation. The symbol $\to_\mu$ will denote the one-step reduction relation of $\lambda\mu$; and the symbols $\twoheadrightarrow_\mu$ and $=_\mu$ will stand for the reflexive, transitive closure and the reflexive, transitive, symmetric closure of the one-step reduction respectively. Finally, when a judgement of the form $M\;: \vdash A$ is derivable according to the typing rules of $\lambda\mu$, we will write $\vdash_\mu M : A$.

4 Translation of $\lambda\mu$ into λC

In this section we give a homomorphic translation of $\lambda\mu$ into λC that preserves typing and reduction.

For the purpose of this translation, we consider that all the variables (i.e., the μ-variables and the λ-variables) belong to the same alphabet.

Definition 4.1 (C-transform) *The C-transform $\langle M\rangle$ of a $\lambda\mu$-term M is inductively defined as follows:*

(i) $\langle x\rangle \;=\; x$;

(ii) $\langle \lambda x.\, M\rangle \;=\; \lambda x.\, \langle M\rangle$;

(iii) $\langle M\, N\rangle \;=\; \langle M\rangle\, \langle N\rangle$;

(iv) $\langle \mu\alpha.\, M\rangle \;=\; \mathcal{C}\,(\lambda\alpha.\, \langle M\rangle)$;

(v) $\langle [\,\alpha\,]\, M\rangle \;=\; \alpha\, \langle M\rangle$.

The two next propositions establish that the C-transform preserves typing and reduction.

Proposition 4.2 *If M is a $\lambda\mu$-term and A is a simple type such that $\vdash_\mu \; M : A$ then $\vdash_C \; \langle M\rangle : A$.*

Proof. We interpret any sequent

$$M \;:\; \Gamma \vdash A,\, \Delta$$

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

$$\Gamma,\, \neg\Delta \vdash M : A,$$

where the context $\neg\Delta$ corresponds to the context Δ in which each type has been negated.

Then, using Felleisen's operator $\mathcal{C}$, we may simulate μ-abstraction and naming as follows:

$$\dfrac{\dfrac{\Gamma,\, \neg\Delta,\, \neg A^{\alpha} \vdash M : \bot}{\Gamma,\, \neg\Delta \vdash \lambda\alpha.\, M : \neg\neg A}}{\Gamma,\, \neg\Delta \vdash \mathcal{C}\,(\lambda\alpha.\, M) : A}$$

$$\dfrac{\neg A^{\alpha} \vdash \alpha : \neg A \qquad \Gamma,\, \neg\Delta \vdash M : A}{\Gamma,\, \neg\Delta,\, \neg A^{\alpha} \vdash \alpha\, M : \bot}$$

□

Proposition 4.3 *Let M and N be $\lambda\mu$-terms. If $M \twoheadrightarrow_\mu N$, then $\langle M\rangle \twoheadrightarrow_c \langle N\rangle$.*

Proof. The property is obvious for the β-reduction steps. For the structural reduction steps, we have that:

$$\begin{aligned}\langle(\mu\alpha.\, M)\, N\rangle &= \mathcal{C}\,(\lambda\alpha.\,\langle M\rangle)\,\langle N\rangle\\ &\to_c \mathcal{C}\,(\lambda k.\,(\lambda\alpha.\,\langle M\rangle)\,(\lambda f.\,k\,(f\,\langle N\rangle)))\\ &\to_c \mathcal{C}\,(\lambda k.\,\langle M\rangle[\lambda f.\,k\,(f\,\langle N\rangle)/\alpha])\end{aligned}$$

Then, to establish that

$$\mathcal{C}\,(\lambda k.\,\langle M\rangle[\lambda f.\,k\,(f\,\langle N\rangle)/\alpha]) \twoheadrightarrow_c \langle\mu\alpha.\, M[N/\!*\,\alpha]\rangle,$$

it remains to establish that

$$\langle M\rangle[\lambda f.\,k\,(f\,\langle N\rangle)/\alpha] \twoheadrightarrow_c \langle M[N/\!*\,\alpha]\rangle[k/\alpha].$$

This last property may be established by induction on the definition of the structural substitution. The only intersting case is when M is of the form $[\alpha]\,O$:

$$\begin{aligned}\langle[\alpha]\,O\rangle[\lambda f.\,k\,(f\,\langle N\rangle)/\alpha] &= (\alpha\,\langle O\rangle)[\lambda f.\,k\,(f\,\langle N\rangle)/\alpha]\\ &= (\lambda f.\,k\,(f\,\langle N\rangle))\,\langle O\rangle[(\lambda f.\,k\,(f\,\langle N\rangle))/\alpha]\\ &\to_c k\,(\langle O\rangle[(\lambda f.\,k\,(f\,\langle N\rangle))/\alpha]\,\langle N\rangle)\\ &\twoheadrightarrow_c k\,(\langle O[N/\!*\,\alpha]\rangle[k/\alpha]\,\langle N\rangle)\\ &\qquad\qquad\qquad\text{by induction hypothesis}\\ &= \alpha\,(\langle O[N/\!*\,\alpha]\rangle\,\langle N\rangle)[k/\alpha]\\ &\qquad\qquad\qquad\alpha \notin FV(\langle N\rangle)\\ &= \langle[\alpha]\,(O[N/\!*\,\alpha]\,N)\rangle[k/\alpha]\\ &= \langle([\alpha]\,O)[N/\!*\,\alpha]\rangle[k/\alpha]\end{aligned}$$

□

Notice that the notion of reduction $\mathcal{C}_{\text{top}}$ does not play any role in the above proof. This notion of reduction is necessary only for the results of the next section.

5 The Inverse Translation

We have shown that $\lambda\mu$ may be injected into λC. In this section, we show that the two calculi are actually isomorphic. To this end, we construct the inverse translation.

The key of this inverse translation is the following $\lambda\mu$-derivation that corresponds to the double negation rule[2]:

$$\dfrac{M \,:\, \vdash \neg\neg A \qquad \dfrac{\dfrac{f \,:\, A^f \vdash A}{[\,\alpha\,]\,f \,:\, A^f \vdash A^\alpha}}{\lambda f.\,[\,\alpha\,]\,f \,:\, \vdash \neg A,\, A^\alpha}}{\dfrac{M\,(\lambda f.\,[\,\alpha\,]\,f) \,:\, \vdash A^\alpha}{\mu\alpha.\,M\,(\lambda f.\,[\,\alpha\,]\,f) \,:\, \vdash A}}$$

This derivation motivates the definition that follows.

Definition 5.1 (μ-transform) *The μ-transform $\overline{M}$ of a $\lambda\mu$-term M is inductively defined as follows:*

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

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

(iii) $\overline{M\,N} = \overline{M}\,\overline{N}$;

(iv) $\overline{\mathcal{C}\,M} = \mu\alpha.\,\overline{M}\,(\lambda f.\,[\,\alpha\,]\,f)$.

By construction, we have that the μ-transform preserves typing.

Proposition 5.2 *If M is a λ_c-term and A is a simple type such that $\vdash_C M : A$ then $\vdash_\mu \overline{M} : A$.*

Proof. A straightforward induction on the derivation of $\vdash_C M : A$. □

The μ-transform does not preserve reduction. This is due to the fact that the elementary reduction steps of λC are more basic than the notion of structural reduction of $\lambda\mu$. Nevertheless, the equality is preserved.

Proposition 5.3 *Let M and N be λ_c-terms. If $M =_c N$, then $\overline{M} =_\mu \overline{N}$.*

Proof. The property is obvious for the β-reduction steps. For the notions of reduction $\mathcal{C}_{\mathrm{L}}$ and $\mathcal{C}_{\mathrm{top}}$ we proceed, respectively, as follows.

$$\begin{aligned}
\overline{(\mathcal{C}\,M)\,N} &= (\mu\alpha.\,\overline{M}\,(\lambda f.\,[\,\alpha\,]\,f))\,\overline{N} \\
&=_\mu \mu\alpha.\,\overline{M}\,(\lambda f.\,[\,\alpha\,]\,(f\,\overline{N})) \\
&=_\mu \mu\alpha.\,\overline{M}\,(\lambda f.\,(\lambda x.\,[\,\alpha\,]\,x)\,(f\,\overline{N})) \\
&=_\mu \mu\alpha.\,(\lambda k.\,\overline{M}\,(\lambda f.\,k\,(f\,\overline{N})))\,(\lambda x.\,[\,\alpha\,]\,x) \\
&= \overline{\mathcal{C}\,(\lambda k.\,M\,(\lambda f.\,k\,(f\,N)))}
\end{aligned}$$

$$\begin{aligned}
\overline{\mathcal{C}\,M} &= \mu\alpha.\,\overline{M}\,(\lambda f.\,[\,\alpha\,]\,f) \\
&=_\mu \mu\alpha.\,\overline{M}\,(\lambda f.\,(\lambda x.\,[\,\alpha\,]\,x)\,f) \\
&=_\mu \mu\alpha.\,(\lambda k.\,\overline{M}\,(\lambda f.\,k\,f))\,(\lambda x.\,[\,\alpha\,]\,x) \\
&= \overline{\mathcal{C}\,(\lambda k.\,M\,(\lambda f.\,k\,f))}
\end{aligned}$$

□

[2] Remark that the resulting $\lambda\mu$-term is not the one given by M. Parigot in [15]. The difference, which is necessary for our purpose, is due to the rules that we have given to handle the negation

It remains to establish that the two transforms are inverse of each other. This is where the notion of reduction $\mathcal{C}_{\text{top}}$ is needed.

Proposition 5.4 *Let M be a closed $\lambda\mu$-term and N be a λ_c-term. The C- and the μ-transforms are such that:*

$$\overline{\langle M \rangle} =_\mu M \tag{a}$$

$$\langle \overline{N} \rangle =_c N \tag{b}$$

Proof. (a) The proof is by induction on the structure of M. We prove that for any $\lambda\mu$-term M

$$\overline{\langle M \rangle} =_\mu M^*,$$

where M^* is obtained by replacing each free μ-variable α occurring in a subterm $[\alpha]\, O$ of M, by the application $\alpha\, O$.

The only non-trivial case is when M is a μ-abstraction:

$$\begin{aligned}
\overline{\langle \mu\alpha.\, O \rangle} &= \overline{\mathcal{C}\,(\lambda\alpha.\, \langle O \rangle)} \\
&= \mu\beta.\,(\lambda\alpha.\, \overline{\langle O \rangle})\,(\lambda f.\,[\beta]\, f) \\
&=_\mu \mu\beta.\,(\lambda\alpha.\, O^*)\,(\lambda f.\,[\beta]\, f) \\
&=_\mu \mu\beta.\, O^*[\lambda f.\,[\beta]\, f/\alpha] \\
&=_\mu (\mu\beta.\, O[\beta/\alpha])^*
\end{aligned}$$

(b) The proof is by induction on the structure of N. The only interesting case is the one of a $\mathcal{C}$-application:

$$\begin{aligned}
\langle \overline{\mathcal{C}\, O} \rangle &= \langle \mu\alpha.\, \overline{O}\,(\lambda f.\,[\alpha]\, f) \rangle \\
&= \mathcal{C}\,(\lambda\alpha.\, \langle \overline{O} \rangle\,(\lambda f.\, \alpha\, f)) \\
&=_c \mathcal{C}\,(\lambda\alpha.\, O\,(\lambda f.\, \alpha\, f)) \\
&=_c \mathcal{C}\, O
\end{aligned}$$

□

We are now in the position of establishing that the two calculi are isomorphic.

Proposition 5.5 *If M and M' are closed $\lambda\mu$-terms, if N and N' are λ_c-terms, then*

$$M =_\mu M' \quad \textit{iff} \quad \langle M \rangle =_c \langle M' \rangle \tag{a}$$

$$N =_c N' \quad \textit{iff} \quad \overline{N} =_\mu \overline{N'} \tag{b}$$

Proof. From Propositions 4.3, 5.3, and 5.4. □

6 Conclusions

The isomorphism that we have presented in this paper clarifies the relation existing between two different calculi that both extend the formulae-as-types principle to classical logic. Surprisingly enough, we have shown that the main differences between the two approaches are primarily syntactic and that first-order $\lambda\mu$-calculus may be seen as a subtheory of a call-by-name variant of Felleisen's syntactic theory of sequential control.

It could be argued, however that our result is only partial because we did not take into account some features of the original $\lambda\mu$ and λC, respectively renaming and the use of the abort operator. As we said, from a proof-theoretic point of view, these two features are merely trivial. Nevertheless, it is interesting to see how we could allow for them.

As for the abort operator, we can modify the definition of the C-transform as follows:

(iv) $\langle [\alpha]\, M \rangle \;=\; \mathcal{A}\,(\alpha\,\langle M \rangle)$,

the other clauses being unchanged. Then, in order to preserve Proposition 4.3, we need the following reduction rule:

$$\mathcal{A}\,(\mathcal{A}\,M) \rightarrow_c \mathcal{A}\,M,$$

which is a particular case of Felleisen's notion of reduction $\mathcal{C}_{\text{idem}}$. On the other hand, the definition of the μ-transform may be kept unchanged but, in order to preserve Proposition 5.3, we would need to allow for the following notion of reduction:

$$\mu\alpha.\, M \rightarrow_\mu M$$

where $\mu\alpha.\, M$ and M are both of type $\perp$, and where α does not occur free in M. This notion of reduction, which is not included in Parigot's theory, is related to our treatment of negation.

As for renaming, it can be simulated in Felleisen's calculus by using the notion of reduction that follows:

$$M\,(\mathcal{C}\,N) \rightarrow_c N\,(\lambda x.\, M\,x),$$

where the type of M is of the form $\neg A$. This notion of reduction, which is absent from Felleisen's theory, is used by F. Barbanera and S. Berardi in [1].

In establishing our isomorphism, we did not use Felleisen's notion of reduction $\mathcal{C}_{\text{R}}$:

$$M\,(\mathcal{C}\,N) \rightarrow_c \mathcal{C}\,(\lambda k.\, N\,(\lambda v.\, \mathcal{A}\,(k\,(M\,v)))).$$

This notion of reduction corresponds, at the level of the $\lambda\mu$-calculus, to the following reduction rule:

$$M\,(\mu\alpha.\, N) \rightarrow_\mu \mu\alpha.\, N[M\,{*}/\alpha],$$

where $N[M\,{*}/\alpha]$ is obtained by replacing inductively each subterm of N of the form $[\alpha]\, O$ by $[\alpha]\,(M\,O)$. This rule is symmetrical to the structural reduction rule. As observed by M. Parigot in [15], to add this rule to his theory destroys the confluence of the calculus. Nevertheless the addition of such rules allows the calculus to have a stronger notion of normal form, and the confluence can be restore by defining an appropriate call-by-value strategy. The resulting calculus provides an alternative solution to the problem of the *uniqueness of the representation of data*. This problem is solved by M. Parigot in [16] by using Krivine's storage operators, which is actually a way of enforcing a call-by-value mechanism.

References

1. 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.
2. 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 on Typed Lambda Calculi and Applications*, pages 45–59. Lecture Notes in Computer Science, 664, Springer Verlag, 1993.
3. H.P. Barendregt. *The lambda calculus, its syntax and semantics.* North-Holland, revised edition, 1984.
4. Ph. de Groote. A CPS-translation of the $\lambda\mu$-calculus. In *Proceedings of the Colloquium on Trees in Algebra and Programming (CAAP'94)*. Lecture Notes in Computer Science, Springer Verlag, 1994.
5. M. Felleisen, D.P. Friedman, E. Kohlbecker, and B. Duba. A syntactic theory of sequential control. *Theoretical Computer Science*, 52:205–237, 1987.
6. M. Felleisen and R. Hieb. The revised report on the syntactic theory of sequential control and state. *Theoretical Computer Science*, 102:235–271, 1992.
7. J.-Y. Girard. A new constructive logic: Classical logic. *Mathematical Structures in Computer Science*, 1:255–296, 1991.
8. J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1989.
9. 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.
10. J.R. Hindley and J.P. Seldin. *Introduction to combinators and λ-calculus.* London Mathematical Society Student Texts. Cambridge University Press, 1986.
11. J.-L. Krivine. *Lambda-calcul, types et modèles.* Masson, 1990.
12. J. Lambek and P.J. Scott. *An introduction to higher order categorical logic.* Cambridge University Press, 1986.
13. 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.
14. 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.
15. 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.

16. 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.
17. 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.
18. D. Prawitz. *Natural Deduction, A Proof-Theoretical Study.* Almqvist & Wiksell, Stockholm, 1965.
19. S. Stenlund. *Combinators λ-terms and proof theory.* D. Reidel Publishing Company, 1972.