On the relations between the syntactic theories of lambda-mu-calculi.

Alexis Saurin

INRIA-Futurs & École polytechnique (LIX)
saurin@lix.polytechnique.fr

Abstract. Since Parigot's seminal article on "an algorithmic interpretation of classical natural deduction" [13], $\lambda\mu$-calculus has been extensively studied both as a typed and an untyped language. Among the studies about the call-by-name lambda-mu-calculus authors used different presentations of the calculus that were usually considered as equivalent from the computational point of view. In particular, most of the papers use one of three variants of the calculus initially introduced by Parigot: *(i)* Parigot's syntax, *(ii)* an extended calculus that satisfies Böhm theorem and *(iii)* a second variant by de Groote he considered when designing an abstract machine for $\lambda\mu$-calculus that contains one more reduction rule. In a previous work [18] we showed that contrarily to Parigot's calculus that does not enjoy separation property as shown by David and Py [3], de Groote's initial calculus, that we refer to as $\Lambda\mu$-calculus, does enjoy the separation property. This evidence the fact that the calculi are really different and suggest that the relationships between the $\lambda\mu$-calculi should be made clear. This is the purpose of the present work.
We first introduce four variants of call-by-name $\lambda\mu$-calculus and then establish some results about the reductions in $\Lambda\mu$-calculus and then investigate the relationships between the $\lambda\mu$-calculi. We finally introduce a new type system for $\Lambda\mu$-calculus and prove subject reduction and strong normalization.

1 Introduction

Curry-Howard in classical logic. Curry-Howard correspondence was first designed as a correspondence between intuitionistic natural deduction and simply typed λ-calculus. Extending the correspondence to classical logic resulted in strong connections with control operators in functional programming languages as first noticed by Griffin [7]. In particular, $\lambda\mu$-calculus [13] was introduced by Michel Parigot as an extension of λ-calculus isomorphic to an alternative presentation of classical natural deduction (known as free deduction) in which one can encode usual control operators and in particular the `call/cc` operator.

Variants of $\lambda\mu$-calculi. Several variants of Parigot's $\lambda\mu$-calculus are considered in the literature. However, their relationship is usually not made clear. However, semantics of those languages can differ as well as their syntactical properties,

for instance, $\Lambda\mu$-calculus satisfies Böhm theorem while $\lambda\mu\eta$ does not. Also, it is sometimes not very clear what properties are true in which variant of $\lambda\mu$-calculus. One of the purposes of this paper is to study and compare different variants of call-by-name $\lambda\mu$-calculus in order to make clear what are the important differences between the calculi and what are their relationships.

The point we want to make in this paper is not that a calculus would be better than another but that they share some properties but differ on some others, affecting the expressiveness and semantics of those calculi.

$\lambda\mu$-calculus and Separation. $\lambda\mu$-calculus became one of the most standard ways to study classical lambda-calculi. As a result, the calculus has been more and more studied and more fundamental questions arose. Among them, knowing whether separation property (also called Böhm theorem [2, 10]) holds for $\lambda\mu$-calculus was one of the important question in the study of $\lambda\mu$-calculus, since separation is a very fundamental property that relates syntax and semantics in a very delicate way. In 2001, David & Py proved that separation fails in $\lambda\mu$-calculus (more precisely the calculus we shall call $\lambda\mu\eta$) by exhibiting a counter-example to separation [3]. In a previous work of 2005, we introduced an extension to $\lambda\mu$-calculus, $\Lambda\mu$-calculus, for which we proved that separation holds [18]. We will further develop the meta-theory of $\Lambda\mu$-calculus in this paper by proving its confluence and by characterizing more precisely the canonical normal forms of $\Lambda\mu$-calculus which are the "values" that we shall separate. Moreover, we introduce a type system for $\Lambda\mu$-calculus which has the property of allowing more terms to be typed while keeping subject reduction and strong normalization.

We regard the type system we consider in the present paper as a first step towards the study a typed separation [20, 10, 6] theorem for $\Lambda\mu$-calculus that we leave for future work. In particular $\Lambda_{\mathcal{S}}$ was designed in order to be a candidate for typed separation.

Structure of the paper. First we present the four variants of CBN $\lambda\mu$-calculus that we shall study in this paper. We then prove confluence of $\Lambda\mu$-calculus in Section 3 and then compare the equational theories of the $\lambda\mu$-calculi in Section 4. Then we introduce in Section 5 a type system for $\Lambda\mu$-calculus, $\Lambda_{\mathcal{S}}$, and prove subject reduction and strong normalization of simply typed $\Lambda\mu$-calculus.

2 Four $\lambda\mu$-calculi

In this section, we present the four variants of call-by-name $\lambda\mu$-calculus that we shall study in this paper. We shall use in this paper an alternative notation for $\lambda\mu$-terms that we introduced and justified in a previous work [18], writing $(t)\alpha$ instead of $[\alpha]t$.

2.1 Parigot's original calculus: $\lambda\mu$.

In 1992, Michel Parigot introduced $\lambda\mu$, an extension of λ-calculus providing "an algorithmic interpretation of classical natural deduction" by allowing for a

$$\frac{}{\Gamma, x : \mathcal{T} \vdash_{\lambda\mu} x \ : \mathcal{T} | \Delta} \ Var_{\mathcal{T}} \qquad \frac{\Gamma \vdash_{\lambda\mu} t \ : B | \Delta, \alpha : A}{\Gamma \vdash_{\lambda\mu} \mu\alpha^{A}.(t)\beta \ : A | (\Delta, \beta : B) \setminus \alpha : A} \ \mu$$

$$\frac{\Gamma, x : \mathcal{T} \vdash_{\lambda\mu} t \ : \mathcal{T}' | \Delta}{\Gamma \vdash_{\lambda\mu} \lambda x^{\mathcal{T}}.t \ : \mathcal{T} \to \mathcal{T}' | \Delta} \ \lambda\text{-Abs} \qquad \frac{\Gamma \vdash_{\lambda\mu} t \ : \mathcal{T} \to \mathcal{T}' | \Delta \qquad \Gamma \vdash_{\lambda\mu} u \ : \mathcal{T} | \Delta}{\Gamma \vdash_{\lambda\mu} (t)u \ : \mathcal{T}' | \Delta} \ \lambda\text{-App}$$

Fig. 1. Type system for $\lambda\mu$-calculus ($\mathcal{T}, A, B$ are the usual simple types with $\to$).

proof-program correspondence *à la* Curry-Howard [9] between $\lambda\mu$-calculus and classical natural deduction.

Definition 1 ($\Sigma_{\lambda\mu}$). *The terms of Parigot's $\lambda\mu$-calculus are defined by the following syntax:*

$$t ::= \ x \mid \lambda x.t \mid (t)u \mid \mu\alpha.(t)\beta$$

with $x \in \mathcal{V}$ and $\alpha, \beta \in \mathcal{V}_c$, $\mathcal{V}$ and $\mathcal{V}_c$ being two disjoint infinite sets of variables. The set of terms of Parigot's $\lambda\mu$-calculus is noted $\Sigma_{\lambda\mu}$. In $\mu\alpha.(t)\beta$, variable β may be bound by $\mu\alpha$.

Definition 2 ($\lambda\mu$-calculus reductions). *$\lambda\mu$-calculus reduction, written $\longrightarrow_{\lambda\mu}$, is induced by the following four reduction rules:*

- $(\lambda x.t)u \longrightarrow_{\beta} t\{u/x\}$
- $(\mu\alpha.n)u \longrightarrow_{\mu} \mu\alpha.n\{(v)u\alpha/(v)\alpha\}$
- $(\mu\alpha.n)\beta \longrightarrow_{\rho} n\{\beta/\alpha\}$
- $\mu\alpha.(t)\alpha \longrightarrow_{\theta} t$ *if* $\alpha \notin FV(t)$

Notice that in reduction ρ, $(\mu\alpha.n)\beta$ is not an element of $\Sigma_{\lambda\mu}$, but what is usually called a named term and is generically written n. Substitution $n\{(v)u\alpha/(v)\alpha\}$ refers to the capture avoiding substitution of any subterm $(v)\alpha$ of n by $(v)u\alpha$.

$\lambda\mu$-calculus satisfies lots of good properties of λ-calculus: confluence of the untyped calculus [13, 17, 3], subject reduction [13] and strong normalization [15, 16] of the typed calculus (see figure 1) being the most central ones.

2.2 $\lambda\mu$-calculus with extensionality: Py's $\lambda\mu\eta$.

Whereas the critical pair between β and η does not prevent Church-Rosser to hold, the critical pair μ/η does not converge:

$$\mu\alpha.n \longleftarrow_{\eta} \lambda x.(\mu\alpha.n)x \longrightarrow_{\mu} \lambda x.\mu\alpha.n\{(v)x\alpha/(v)\alpha\}$$

This is probably why η is not considered in Parigot's original presentation. In his PhD, Walter Py studied confluence properties of $\lambda\mu$-calculus and was interested in Böhm theorem for $\lambda\mu$-calculus. In studying separation, it is needed to have extensionality so that Py added η to $\lambda\mu$-calculus and restored separation with an additional rule, ν, that makes the above diagram to converge.

$\lambda\mu\eta$-calculus is obtained by adding to $\lambda\mu$-calculus the following two rules:

– $\lambda x.(t)x \longrightarrow_\eta t$ if $x \notin FV(t)$
– $\mu\alpha.n \longrightarrow_\nu \lambda x.\mu\alpha.n\,\{(v)x\alpha/(v)\alpha\}$ if $x \notin FV(n)$

David & Py proved that separation fails in $\lambda\mu\eta$-calculus [17, 3] by exhibiting a counter-example to separation[1]: $w_0, w_1 \in \Sigma_{\lambda\mu}$ that are solvable, not equivalent for the equivalence relation induced by $\lambda\mu\eta$-reduction rules but that no context can distinguish. In a previous work [18], we defined $\Lambda\mu$, an extension to $\lambda\mu\eta$ for which we proved Böhm theorem.

2.3 A $\lambda\mu$-calculus satisfying Böhm theorem: $\Lambda\mu$-calculus.

Definition 3 ($\Sigma_{\Lambda\mu}$). *$\Lambda\mu$-terms are defined by the following syntax:*

$$t ::= \; x \mid \lambda x.t \mid (t)u \mid \mu\alpha.t \mid (t)\alpha$$

where x ranges over a set $\mathcal{V}_t$ of term variables and α ranges over a set $\mathcal{V}_s$ of stream variables.

Notice that $\Sigma_{\lambda\mu} \subset \Sigma_{\Lambda\mu}$ and that in $\Lambda\mu$-calculus, named terms are usual terms. In $\Lambda\mu$-calculus, it is possible to consider terms such as $\mu\alpha.\mu\beta.t$ or $\lambda x.(t)\alpha y$.

Definition 4 ($\Lambda\mu$**-calculus reductions).** *$\Lambda\mu$-calculus reduction, written $\longrightarrow_{\Lambda\mu}$, is induced by the following five reduction rules[2]:*

– $(\lambda x.t)u \longrightarrow_{\beta_T} t\,\{u/x\}$
– $\lambda x.(t)x \longrightarrow_{\eta_T} t$ *if* $x \notin FV(t)$
– $(\mu\alpha.t)\beta \longrightarrow_{\beta_S} t\,\{\beta/\alpha\}$
– $\mu\alpha.(t)\alpha \longrightarrow_{\eta_S} t$ *if* $\alpha \notin FV(t)$
– $\mu\alpha.t \longrightarrow_{fst} \lambda x.\mu\alpha.t\,\{(v)x\alpha/(v)\alpha\}$ *if* $x \notin FV(t)$

μ-reduction can be added to $\Lambda\mu$-calculus reduction but this does not change anything since μ-reduction can be simulated by a fst-reduction followed by a β-reduction: $(\mu\alpha.t)u \longrightarrow_{fst} (\lambda x.\mu\alpha.t\,\{(v)x\alpha/(v)\alpha\})u \longrightarrow_\beta \mu\alpha.t\,\{(v)u\alpha/(v)\alpha\}$.

A separation result is stated with respect to a set of observables (the $\beta\eta$-normal forms in λ-calculus). Since *fst* is an expansion rule, there are very few normal forms in $\Lambda\mu$. We thus consider a set of canonical normal forms [3, 18]:

Definition 5. *A $\Lambda\mu$-term t is in* **canonical normal form (CNF)** *if it is $\beta_T\eta_T\beta_S\eta_S$-normal and it contains no subterm $(\lambda x.u)\alpha$ nor $(\mu\alpha.u)v$.*

Theorem 1 (Böhm theorem for $\Lambda\mu$-calculus [18]). *If t and t' are two non $\beta_T\eta_T\beta_S\eta_S$fst-equivalent closed canonical normal forms, there exists a context[3] $C[]$ such that $C[t] \to^\star_{\Lambda\mu} \lambda x.\lambda y.x$ and $C[t'] \to^\star_{\Lambda\mu} \lambda x.\lambda y.y$.*

---

[1] w_0, w_1 are obtained by substituting y by $0 = \lambda x, x'.x'$ and $1 = \lambda x, x'.x$ respectively in $w = \lambda x.\mu\alpha.((x)\mu\beta.(x)u_0y\alpha)u_0\alpha$ with $u_0 = \mu\delta.(\lambda z_1, z_2.z_2)\alpha$.

[2] β_T, η_T, β_S, η_S, *fst* correspond respectively to $\beta, \eta, \rho, \theta$ and ν; see [18] for details.

[3] A context that may be "stream applicative": $[]t_1^1 \dots t_{n_1}^1\alpha_1 \dots t_1^k \dots t_{n_k}^k\alpha_k$.

Remark 1. In [4], de Groote introduced an extension to $\lambda\mu$-calculus, that we shall refer to as $\lambda\mu_{dG}$ and that is close to $\Lambda\mu$-calculus except it has neither η_T nor *fst*. In [11], Ong considers a calculus built on $\Sigma_{\Lambda\mu}$ which is very close to $\Lambda\mu$ but except that it is presented as an equational theory. $\lambda\mu_{dG}$ can be considered as a subcalculus $\Lambda\mu$, in the following, we will not consider much this calculus.

2.4 de Groote's extended syntax with ϵ-reduction: $\lambda\mu\epsilon$

Philippe de Groote, while building an abstract machine for $\lambda\mu$-calculus [5], considered another extension to Parigot's $\lambda\mu$, also based on terms of $\Sigma_{\Lambda\mu}$: the $\lambda\mu\epsilon$-calculus. $\lambda\mu\epsilon$ has an additional rule, ϵ-reduction: $\mu\alpha.\mu\beta.t \to_\epsilon \mu\alpha.\,|t|_\beta$ where $|t|_\beta$ is the result of removing all the free occurrences of β in t. A constraint on reduction ρ is added by de Groote not to lose confluence: $\mu\gamma.(\mu\alpha.t)\beta \longrightarrow_\rho \mu\gamma.t\,\{\beta/\alpha\}$. Otherwise there is a critical pair:

$$|t|_\alpha\,\{\delta/\gamma\}\,\{\zeta/\beta\} \leftarrow_{\rho;\rho;\epsilon} (\mu\gamma.\mu\beta.\mu\alpha.t)\delta\zeta \to_{\rho;\rho;\epsilon} |t|_\beta\,\{\delta/\gamma\}\,\{\zeta/\alpha\}$$

η-reduction cannot be added to $\lambda\mu\epsilon$ or confluence is lost[4]:

$$\mu\alpha.\lambda x.\mu\beta.y \leftarrow_\mu \mu\alpha.\lambda x.(\mu\beta.y)x \to_\eta \mu\alpha.\mu\beta.y \to_\epsilon \mu\alpha.y$$

Definition 6 ($\lambda\mu\epsilon$-calculus reductions). *$\lambda\mu\epsilon$-calculus reduction, written $\longrightarrow_{\lambda\mu\epsilon}$, is induced by the following five reduction rules:*

- $(\lambda x.t)u \longrightarrow_\beta t\,\{u/x\}$
- $(\mu\alpha.t)u \longrightarrow_\mu \mu\alpha.t\,\{(v)u\alpha/(v)\alpha\}$
- $\mu\gamma.(\mu\alpha.t)\beta \longrightarrow_\rho \mu\gamma.t\,\{\beta/\alpha\}$
- $\mu\alpha.(t)\alpha \longrightarrow_\theta t$ *if* $\alpha \notin FV(t)$
- $\mu\alpha.\mu\beta.t \to_\epsilon \mu\alpha.\,|t|_\beta$

In the rest of this paper, we shall prove some new results about $\Lambda\mu$-calculus and prove some properties relating the four calculi introduced in this section. We already state the following:

Lemma 1. *$\Sigma_{\lambda\mu}$ and $\lambda\mu$-calculus are stable by $\Lambda\mu$-reductions and $\lambda\mu\epsilon$-reductions:*

- *if $t \in \Sigma_{\lambda\mu}$ and $t \to^\star_{\Lambda\mu} u$, then $u \in \Sigma_{\lambda\mu}$ and $t \to^\star_{\lambda\mu\eta} u$;*
- *if $t \in \Sigma_{\lambda\mu}$ and $t \to^\star_{\lambda\mu\epsilon} u$, then $u \in \Sigma_{\lambda\mu}$ and $t \to^\star_{\lambda\mu} u$.*

3 Syntactical results for pure $\Lambda\mu$-calculus

3.1 $\Lambda\mu$-calculus reduction system

Definition 7 (β, β^{var}, η, fst^-).

- *β is the subsystem made of reductions β_T and β_S;*

[4] Many details on problems of confluence in $\lambda\mu\epsilon$ with η can be found in Py's thesis [17].

- $\boldsymbol{\eta}$ *is the subsystem made of reductions* η_T *and* η_S*;*
- $\boldsymbol{\beta fst}$ *is the subsystem* $\beta_T\beta_S fst$ *and* $\boldsymbol{\beta\eta fst}$ *for the full* $\Lambda\mu$*-reduction system;*
- fst^- *is the restriction of* fst *to redexes* $t = \mu\alpha.t'$ *such that* t' *contains at least one subterm* $(\lambda x.u)\alpha$ *or such that* t *is applied to a term.* $\rightarrow_{\Lambda\mu^-}$ *is* $\rightarrow_{\boldsymbol{\beta\eta} fst^-}$*;*
- $\boldsymbol{\beta}^{var}$ *for the subsystem of* $\boldsymbol{\beta}$ *that reduces a* $\boldsymbol{\beta}$*-redex only when the argument is a variable:* $(\lambda x.t)y \rightarrow_{\beta_T^{var}} t\{y/x\}$*.*
- $\sim_{\Lambda\mu}$ *and* $\sim_{\Lambda\mu^-}$ *are the equivalences associated with* $\rightarrow_{\Lambda\mu}$ *and* $\rightarrow_{\Lambda\mu^-}$*.*

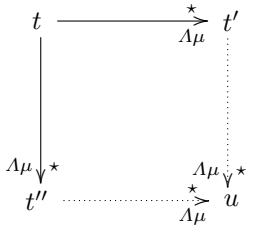
Proposition 1. *μ-closed canonical normal forms are exactly the μ-closed $\Lambda\mu$-terms in $\boldsymbol{\beta\eta} fst^-$-normal form.*

Remark 2. $\sim_{\Lambda\mu}=\sim_{\Lambda\mu^-}$ but we do not consider the reduction system $\Lambda\mu^-$ since it is not confluent. Indeed, there are different canonical normal forms the are $\sim_{\Lambda\mu}$-equivalent but they are normal forms for $\rightarrow_{\Lambda\mu^-}$.

3.2 Confluence of $\Lambda\mu$-calculus

In this section, we prove confluence of $\Lambda\mu$-calculus for μ-closed terms:

Theorem 2. *$\Lambda\mu$-calculus is confluent on μ-closed terms:*
For any t, t', t'' μ-closed $\Lambda\mu$-terms, there exists $u \in \Sigma_{\Lambda\mu}$ such that

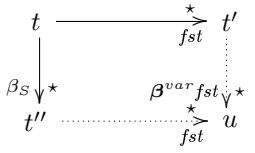


Remark 3. Notice that the hypothesis on μ-closed terms is a necessary restriction considering that the term $(\mu\beta.x)\alpha$ may reduce to x or to $(\lambda y.\mu\gamma.x)\alpha$ which cannot reduce to the same term.

Proof. Confluence of $\Lambda\mu$-calculus is proved thanks to some preliminary lemmas. Essentially, confluence of the calculus follows from the confluence of subsystem $\boldsymbol{\beta fst}$ (proposition 2), confluence of subsystem $\boldsymbol{\eta}$ (proposition 3) and the commutation of the two previous systems (proposition 4): thanks to Hindley-Rosen lemma, this ensures confluence of $\Lambda\mu$-calculus. □

The following lemma is the crucial one:

Lemma 2.



Lemma 3. *$\beta_T fst$ (resp. $\beta^{var} fst$) and βfst commute;*

Proposition 2. *βfst-reduction is confluent.*

Proposition 3. *η-reduction is confluent.*

Proposition 4 is consequence of η commuting with β_T, β_S and *fst* respectively:

Proposition 4. *η commutes with βfst.*

Confluence of $\lambda\mu\eta$-calculus is an easy consequence of confluence of $\Lambda\mu$-calculus:

Corollary 1. *$\lambda\mu\eta$-calculus is confluent on μ-closed terms.*

Proof. Indeed, if $t \in \Sigma_{\lambda\mu}$, μ-closed, is such that $t \to^\star_{\lambda\mu\eta} u, v$ then $t \to^\star_{\Lambda\mu} u, v$ and thus there exists w such that $u, v \to^\star_{\Lambda\mu} w$ but thanks to lemma 1, one has $u, v, w \in \Sigma_{\lambda\mu}$ and $u, v \to^\star_{\lambda\mu\eta} w$. □

Remark 4. A proof of confluence for $\lambda\mu\eta$ can be found in Py's thesis [17] and is outlined in [3]. Our proof of confluence for $\Lambda\mu$-calculus uses some of the ideas of the proof by Py but is simpler, in particular we can avoid a lengthy development where Py uses anontations on terms. The simplification lies in particular in the proof of lemma 2. As a result, the proof we present here is, to our knowledge, the shortest known proof of confluence for $\lambda\mu\eta$.

The following proposition is actually not a trivial consequence of confluence since even though t, u are μ-closed, the sequence of terms justifying $t =_{\Lambda\mu} u$ may involve non-closed terms for which confluence does not hold in general:

Proposition 5. *If $t, u \in \Sigma_{\Lambda\mu}$ are μ-closed and $t =_{\Lambda\mu} u$ then there exists a $v \in \Sigma_{\Lambda\mu}$ such that $t, u \longrightarrow^\star_{\Lambda\mu} v$.*

3.3 Characterizing equivalent canonical normal forms in $\Lambda\mu$.

The following property corresponds, in $\Lambda\mu$-calculus, to the property of uniqueness of the $\beta\eta$-normal form in λ-calculus.

Proposition 6. *Two μ-closed canonical normal forms are $\Lambda\mu$-equivalent if, and only if, they are fst-equivalent:*

$$\textit{if } t, u \in \Sigma_{\Lambda\mu} \textit{ are CNF, then } t =_{\Lambda\mu} u \Leftrightarrow t =_{fst} u.$$

Proof. The proposition is a simple consequence of proposition 5: if $t =_{\Lambda\mu} u$ are μ-closed canonical normal forms, there exists $v \in \Sigma_{\Lambda\mu}$ such that $t, u \to^\star_{\Lambda\mu} v$. Since t, u are CNF, the only possible redexes in t, u are *fst*-redexes and no other redexes will be created by a *fst*-reduction. Thus $t, u \to^\star_{fst} v$ and $t =_{fst} u$. □

4 Comparing the $\lambda\mu$-calculi

4.1 Conservative extensions

Proposition 7 (Conservative extensions).

- *$\Lambda\mu$-calculus is a conservative extension of $\lambda\mu\eta$-calculus: if $t, u \in \Sigma_{\lambda\mu}$ then*

$$t =_{\Lambda\mu} u \Leftrightarrow t =_{\lambda\mu\eta} u.$$

- *$\lambda\mu\epsilon$-calculus is a conservative extension of $\lambda\mu$-calculus: if $t, u \in \Sigma_{\lambda\mu}$ then*

$$t =_{\lambda\mu\epsilon} u \Leftrightarrow t =_{\lambda\mu} u.$$

Proof. It is immediate that if $t, u \in \Sigma_{\lambda\mu}$ then $t =_{\lambda\mu\eta} u$ implies $t =_{\Lambda\mu} u$ and $t =_{\lambda\mu} u$ implies $t =_{\lambda\mu\epsilon} u$. The converse properties require confluence of the calculi and lemma 1: By confluence of $\Lambda\mu$ (resp. $\lambda\mu\epsilon$), if $t =_{\Lambda\mu} u$ (resp. $t =_{\lambda\mu\epsilon} u$) there exists $v \in \Sigma_{\Lambda\mu}$ such that $t \to^\star_{\Lambda\mu} v \leftarrow^\star_{\Lambda\mu} u$ (resp. $t \to^\star_{\lambda\mu\epsilon} v \leftarrow^\star_{\lambda\mu\epsilon} u$). By lemma 1, if moreover $t, u \in \Sigma_{\lambda\mu}$ then there exists $v \in \Sigma_{\lambda\mu}$ such that $t \to^\star_{\lambda\mu\eta} v \leftarrow^\star_{\lambda\mu\eta} u$ (resp. $t \to^\star_{\lambda\mu} v \leftarrow^\star_{\lambda\mu} u$) and finally $t =_{\lambda\mu\eta} u$ (resp. $t =_{\lambda\mu} u$). □

Proposition 8 ($=_{\Lambda\mu}$ and $=_{\lambda\mu\epsilon}$ are incomparable.). *There exist $t, u, v \in \Sigma_{\Lambda\mu}$ such that $t =_{\Lambda\mu} u$ and $t \neq_{\lambda\mu\epsilon} u$ and $t =_{\lambda\mu\epsilon} v$ and $t \neq_{\Lambda\mu} v$.*

Proof. Let $t \in \Sigma_{\Lambda\mu}$ be a term of the form $\mu\alpha.t'$ in canonical normal form with no two consecutive μ-abstractions. Let $u = \lambda x.\mu\alpha.t'\{(u)x\alpha/(u)\alpha\}$ and $v = \mu\alpha.\mu\beta.t'$. Then one has $t =_{fst} u$ and $t =_\epsilon v$ so that $t =_{\Lambda\mu} u$ and $t =_{\lambda\mu\epsilon} v$.

t is a $\lambda\mu\epsilon$-normal form and so is u (variable x cannot create a $\lambda\mu\epsilon$-redex without contradicting that t is a CNF): they are distinct normal form of $\lambda\mu\epsilon$-calculus and thus, by confluence of $\lambda\mu\epsilon$, $t \neq_{\lambda\mu\epsilon} u$. On the other hand, t and v are in CNF so that $t =_{\Lambda\mu} v$ if and only if $t =_{fst} v$. But t and v contain a different number of μ-abstraction although *fst*-reduction preserves the number of μ. As a conclusion $t \neq_{fst} v$ and finally $t \neq_{\Lambda\mu} v$. □

4.2 Separability properties

We showed in a previous work [18] that $\Lambda\mu$-calculus had the separation property: two canonical normal forms are equivalent if and only if they cannot be separated by any context. On the other hand, it is known that $\lambda\mu\eta$ and a fortiori $\lambda\mu$ do not satisfy separability [17, 3].

What can we say about separability in $\lambda\mu\epsilon$? Stating separation in $\lambda\mu\epsilon$-calculus would require to consider the classes modulo $\lambda\mu\epsilon$-rules plus η. However, this makes the study very complex since $\lambda\mu\epsilon + \eta$ is not confluent and thus it is difficult to say any thing about two terms not being equated by the equational theory. We shall simply consider an example of two terms being observationally equivalent whereas they are not equationally equivalent in $\lambda\mu\epsilon$: $\mu\alpha.0$ and $\mu\alpha.1$ are observationally equivalent. Indeed, they cannot be separated by any context:

– $(\mu\alpha.t)u \to_\mu \mu\alpha.t$ if $\alpha \notin FV(t)$.
– $\mu\gamma.(\mu\alpha.t)\beta \to_\rho \mu\gamma.t$ which is α-equivalent to $\mu\alpha.t$.
– $\mu\beta.\mu\alpha.t \to_\epsilon \mu\beta\,|t|_\alpha = \mu\beta.t$ which is α-equivalent to $\mu\alpha.t$.

Actually, any two terms of the form $\mu\alpha.t$ with t a closed term are observationally equivalent to $\mu\alpha.0$.

5 Simply-typed $\Lambda\mu$-calculus

In this section, we introduce a type system for $\Lambda\mu$-calculus and prove some properties about it, namely that it can type strictly more terms than Parigot's type system, that it has subject reduction and strong normalization.

One may of course think of typing $\Lambda\mu$ thanks using a type system for standard classical λ-calculi, similar to the one shown in figure 1, by adding rules:

$$\frac{\Gamma \vdash t \ : \bot | \Delta, \alpha : A}{\Gamma \vdash \mu\alpha^A.t \ : A | \Delta}\ \mu Abs \qquad \frac{\Gamma \vdash t \ : A | \Delta, \alpha : A}{\Gamma \vdash (t)\alpha \ : \bot | \Delta, \alpha : A}\ \mu App$$

The system uses a typing discipline *à la* Church by writing explicitly the type on the abstracted (stream or term) variables. Considering that we have an expansion rule, we restrict the *fst* rule to apply only on terms of type $A \to B$ in order the achieve subject reduction: $\mu\alpha^{A\to B}.t \quad \longrightarrow_{fst} \quad \lambda x^A.\mu\beta^B.t\,\{(u)x\beta/(u)\alpha\}$.

This approach would not be satisfactory since many $\Lambda\mu$-terms would not be typable and in particular terms essential for separation to hold.

Making streams first-class citizens in the typed setting. We look for a type system that would reflect in types the stream construction. In particular, since μ is seen as a stream abstraction, one might think of a functional type for streams: if the term t is of type $\mathcal{T}$ when stream α is of stream type $\mathcal{S}$, then $\mu\alpha.t$ would be of the type of a stream functional from $\mathcal{S}$ to $\mathcal{T}$ (that we write $\mathcal{S} \Rightarrow \mathcal{T}$). We can thus think of the following typing rules for μ-abstracted terms:

$$\frac{\Gamma \vdash t \ : \mathcal{T} | \Delta, \alpha : \mathcal{S}}{\Gamma \vdash \mu\alpha^{\mathcal{S}}.t \ : \mathcal{S} \Rightarrow \mathcal{T} | \Delta}\ Abs_{\mathcal{S}} \qquad \frac{\Gamma \vdash t \ : \mathcal{S} \Rightarrow \mathcal{T} | \Delta, \alpha : \mathcal{S}}{\Gamma \vdash (t)\alpha \ : \mathcal{T} | \Delta, \alpha : \mathcal{S}}\ App_{\mathcal{S}}$$

A type mismatch. Rule *fst* does complicate the definition of a type system for $\Lambda\mu$ that would take streams into account: whereas $\mu\alpha^{\mathcal{S}}.t$ is of a stream type, say $\mathcal{S} \Rightarrow \mathcal{T}$, the term resulting from $\mu\alpha.t$ by applying the *fst* rule once (namely $\lambda x^A.\mu\beta^{\mathcal{S}'}.t\,\{(u)x\beta/(u)\alpha\}$) should be of a standard function type $A \to B$ (more precisely $A \to (\mathcal{S}' \Rightarrow \mathcal{T}')$). Moreover, streams are streams of terms and they should be related, not only by the *fst* rule, but also by allowing to apply a term to a stream functional (for instance $(\mu\alpha.t)u$) and conversely, one might want to apply a stream to a λ-abstracted term (for instance $(\lambda x.t)\alpha$).

$\Rightarrow$-types and $\rightarrow$-types should be related in some way. *fst* gives the key to this connection; we thus analyze more carefully this rule in the following paragraph.

A relation over stream types. *fst* was synthesized in $\Lambda\mu$-calculus (and previously in $\lambda\mu$-calculus by Py [17, 3][5]) as the result of an η-expansion followed by a μ-reduction. In the typed case, the η-expansion can occur only on $\rightarrow$-type terms. This restriction adapted to $\Lambda\mu$-calculus results in the condition that $\mu\alpha.t$ is of a stream type of the form $(\mathcal{T} \rightarrow \mathcal{S}) \Rightarrow \mathcal{T}'$. After an application of *fst*, we have term $\lambda x.\mu\beta.t\,\{(u)x\beta/(u)\alpha\}$ that should be of type $\mathcal{T} \rightarrow (\mathcal{S} \Rightarrow \mathcal{T}')$.

This corresponds to an associativity rule between constructors $\rightarrow$ and $\Rightarrow$:

$$(\mathcal{T} \rightarrow \mathcal{S}) \Rightarrow \mathcal{T}' =_{\text{assoc}_{\Rightarrow}^{\rightarrow}} \mathcal{T} \rightarrow (\mathcal{S} \Rightarrow \mathcal{T}')$$

5.1 Simply typed streams: $\Lambda_{\mathcal{S}}$.

Pre-Types and Types. We now define the type system $\Lambda_{\mathcal{S}}$ for $\Lambda\mu$-calculus.

Definition 8 ($\Lambda_{\mathcal{S}}$ pre-types). *The pre-types are given by the following grammar:*

Term pre-types:	$\mathcal{T}, A, B, \ldots ::= o_i \mid A \rightarrow B \mid \mathcal{S} \Rightarrow \mathcal{T}$
Stream pre-types:	$\mathcal{S}, P, Q, \ldots ::= \sigma_i \mid \mathcal{T} \rightarrow \mathcal{S} \mid \bot$

o_i and σ_i are respectively term and stream type variables. We keep a $\bot$ constant in the calculus, more for tradition than for real need: $\bot$ type may be regarded as a distinguished stream type variable[6]. We might want to withdraw this $\bot$ in future works, however it will be useful when studying the relationships between $\lambda\mu$-typable and $\Lambda_{\mathcal{S}}$-typable terms.

Definition 9 ($\equiv_{fst}$). *$\equiv_{fst}$ is a congruence relation over pre-types which is the symmetric, reflexive and transitive closure of relation $\succ_{fst}$ defined by*

$$(\mathcal{T} \rightarrow \mathcal{S}) \Rightarrow \mathcal{T}' \succ_{fst} \mathcal{T} \rightarrow (\mathcal{S} \Rightarrow \mathcal{T}')$$

Types of $\Lambda_{\mathcal{S}}$ are always considered up to this congruence relation:

Definition 10 ($\Lambda_{\mathcal{S}}$ types). *A $\Lambda_{\mathcal{S}}$ type is an equivalence class for $\equiv_{fst}$.*

Typed $\Lambda\mu$-calculus is considered *à la Church*, that is the syntax of typed $\Lambda\mu$-terms is as follows:

Definition 11 (Typed $\Lambda\mu$-calculus). $t ::= x \mid \lambda x^{\mathcal{T}}.t \mid (t)u \mid \mu\alpha^{\mathcal{S}}.t \mid (t)\alpha$.

We show in figure 2 the type system $\Lambda_{\mathcal{S}}$ for $\Lambda\mu$-calculus. In this type system, we deal with pre-types and an explicit conversion rule between two equivalent pre-types.

---

[5] It had actually been already briefly discussed in 1993 by Parigot [14].

[6] It may alternatively be seen as a variable that cannot be substituted by other types.

$$\frac{}{\Gamma, x:\mathcal{T} \vdash x \ :\mathcal{T}|\Delta}\ Var_{\mathcal{T}} \qquad \frac{\Gamma \vdash t \ :\mathcal{T}|\Delta}{\Gamma \vdash t \ :\mathcal{T}'|\Delta} \equiv_{fst} \quad (\text{provided } \mathcal{T} \equiv_{fst} \mathcal{T}')$$

$$\frac{\Gamma, x:\mathcal{T} \vdash t \ :\mathcal{T}'|\Delta}{\Gamma \vdash \lambda x^{\mathcal{T}}.t \ :\mathcal{T} \to \mathcal{T}'|\Delta}\ Abs_{\mathcal{T}} \qquad \frac{\Gamma \vdash t \ :\mathcal{T} \to \mathcal{T}'|\Delta \qquad \Gamma \vdash u \ :\mathcal{T}|\Delta}{\Gamma \vdash (t)u \ :\mathcal{T}'|\Delta}\ App_{\mathcal{T}}$$

$$\frac{\Gamma \vdash t \ :\mathcal{T}|\Delta, \alpha:\mathcal{S}}{\Gamma \vdash \mu\alpha^{\mathcal{S}}.t \ :\mathcal{S} \Rightarrow \mathcal{T}|\Delta}\ Abs_{\mathcal{S}} \qquad \frac{\Gamma \vdash t \ :\mathcal{S} \Rightarrow \mathcal{T}|\Delta, \alpha:\mathcal{S}}{\Gamma \vdash (t)\alpha \ :\mathcal{T}|\Delta, \alpha:\mathcal{S}}\ App_{\mathcal{S}}$$

Fig. 2. $\Lambda_{\mathcal{S}}$: a type system for $\Lambda\mu$-calculus.

5.2 Typed Reduction Rules.

The *fst* rule is an expansion rule and shall thus be treated with care if one wants subject reduction to hold in $\Lambda_{\mathcal{S}}$. In the typed case, we will require for allowing an application of the *fst* on t that the term has a type represented by a pre-type of shape $(\mathcal{T}_1 \to \mathcal{S}) \Rightarrow \mathcal{T}_2$. This requirement is similar to the condition on the η-expansion application in simply typed λ-calculus and is necessary to satisfy subject reduction in the presence of an expansion rule. The η-expansion is usually restricted as: $t : A \to B \to_{\eta^{\to}_{exp}} \lambda x^A.(t)x : A \to B$.

We require the same sort of constraint on *fst*-rule:

Definition 12 ($fst^{\to}$ reduction). *Reduction $fst^{\to}$ is defined as a restriction on fst-reduction on typed $\Lambda\mu$-terms as follows:*

$$\mu\alpha^{A\to\mathcal{S}}.t \to_{fst^{\to}} \lambda x^A \mu\beta^{\mathcal{S}}.t\,\{(u)x\beta/(u)\alpha\}$$

One can notice that $fst^{\to}$ is an intermediate reduction between *fst* and fst^{-}:

Proposition 9. *Let t, u be two typed $\Lambda\mu$-terms. The following implications hold:*

$$t \to_{fst^{-}} u \quad \Rightarrow \quad t \to_{fst^{\to}} u \quad \Rightarrow \quad t \to_{fst} u$$

The converse implications do not hold.

5.3 Comments about the type system $\Lambda_{\mathcal{S}}$.

Moving from $\Rightarrow$ to $\parr$. Contrarily to what the notation $\Rightarrow$ may suggest, no duality is involved with this connective. The rule $Abs_{\mathcal{S}}$ would rather suggest the $\Rightarrow$ connective to be related with the $\parr$ connective of linear logic. This is precisely what we evidence in a related work with Pagani [12]: when translating $\Lambda_{\mathcal{S}}$ into (a kind of) polarized proof nets, $\mathcal{T}_1 \to \mathcal{T}_2$ becomes as usual $?\mathcal{T}_1^{\perp} \parr \mathcal{T}_2$ while $\mathcal{S} \Rightarrow \mathcal{T}$ is translated into $\mathcal{S} \parr \mathcal{T}$. $\equiv_{fst}$ is thus an associativity property of $\parr$ which is perfectly sound logically: $(?\mathcal{T}^{\perp} \parr \mathcal{S}) \parr \mathcal{T} \equiv_{fst} ?\mathcal{T}^{\perp} \parr (\mathcal{S} \parr \mathcal{T})$.

Relation with $\lambda\mu\widehat{\mathsf{tp}}$ type system. Herbelin et al [1, 8] introduced recently a calculus $\lambda\mu\widehat{\mathsf{tp}}$ which is a $\lambda\mu$-calculus with one dynamically bound variable. This

allows them to model call-by-value and call-by-name delimited continuation. They noticed that call-by-name $\lambda\mu\widehat{\mathsf{tp}}$ is very close to $\Lambda\mu$-calculus and they introduced independently a type system for this calculus which is very similar to $\Lambda_{\mathcal{S}}$. They have however a different structure for typing judgements: $\Gamma \vdash_{\Sigma} M : A; \Delta$ (Σ, annotating the $\vdash$ is a list of types).

In a current work with Herbelin and Ghilezan, we are investigating further the meta-theory $\lambda\mu\widehat{\mathsf{tp}}$ and interestingly $\lambda\mu\epsilon$ appears also to be connected to $\lambda\mu\widehat{\mathsf{tp}}$ when some critical pair naturally arising with the dynamic variable $\widehat{\mathsf{tp}}$ is oriented in the opposite direction: $[\widehat{\mathsf{tp}}]\mu\alpha.c \longrightarrow c\{\widehat{\mathsf{tp}}/\alpha\}$.

5.4 Properties of $\Lambda_{\mathcal{S}}$.

$\Lambda_{\mathcal{S}}$ Types Strictly More Terms Than Parigot's $\lambda\mu$. We show that every typable term of Parigot's $\lambda\mu$-calculus can be typed in $\Lambda_{\mathcal{S}}$.

Theorem 3. *Let t a $\lambda\mu$-term in Parigot's syntax. If there exists Γ, Δ and A such that $\Gamma \vdash_{\lambda\mu} t : A|\Delta$, then there exists $\Gamma^{\mathcal{S}}, \Delta^{\mathcal{S}}$ and $A^{\mathcal{S}}$ such that $\Gamma^{\mathcal{S}} \vdash_{\Lambda_{\mathcal{S}}} t : A^{\mathcal{S}}|\Delta^{\mathcal{S}}$.*

Definition 13. *We consider $o_\perp$ a special term type variable and we define the two following transformations on the types of Parigot's $\lambda\mu$-calculus to $\Lambda_{\mathcal{S}}$ pre-types.*

- *Term pre-types: (i) $(o)^{\mathcal{T}} = (o \to \perp) \Rightarrow o_\perp$ (ii) $(A \to B)^{\mathcal{T}} = A^{\mathcal{T}} \to B^{\mathcal{T}}$*
- *Stream pre-types: (i) $(o)^{\mathcal{S}} = o \to \perp$ (ii) $(A \to B)^{\mathcal{S}} = A^{\mathcal{T}} \to B^{\mathcal{S}}$*

Proposition 10. *Given a simple type A, then $A^{\mathcal{T}} \equiv_{fst} A^{\mathcal{S}} \Rightarrow o_\perp$.*

Remark 5. The previous theorem helps to understand more precisely what is the limitation of Parigot's $\lambda\mu$-calculus with respect to the flexibility of $\Lambda\mu$-calculus: images of $\lambda\mu$-terms need never be assigned a type of shape $\mathcal{S}_1 \Rightarrow (\mathcal{S}_2 \Rightarrow A)$.

Type Preservation. Typed $\Lambda\mu$-calculus satisfies subject reduction:

Theorem 4 (Subject Reduction). *Reduction of typed $\Lambda\mu$-terms preserves type: let $t, u \in \Lambda\mu$. If $\Gamma \vdash t : A|\Delta$ and $t \to_{\Lambda\mu^{\to}} u$ then $\Gamma \vdash u : A|\Delta$.*

Strong Normalization. Finally, we prove strong normalization:

Theorem 5 (Typed $\Lambda\mu$-calculus is strongly normalizing). *Let t be a well-typed term in $\Lambda_{\mathcal{S}}$. There is no infinite reduction from t in $\Lambda\mu^{\to}$.*

The theorem is proved thanks to a method inspired by one of the proofs given by Parigot in [16] for proving strong normalization of simply typed $\lambda\mu$-calculus. It consists in providing a translation of typed $\Lambda\mu$-term into simply typed λ-calculus and deducing strong normalization of typed $\Lambda\mu$-calculus after strong normalization of simply typed λ-calculus.

We first give a translation of $\Lambda_{\mathcal{S}}$ types into simple types:

Definition 14. *To each pre-type of $\Lambda_{\mathcal{S}}$, we associate a simple type as follows: We first enrich the set of type variable of the simply typed λ-calculus: to each stream-type variable σ, one considers a new simple type variable written σ as well. Moreover, we add a new variable $\sigma_\perp$.*

- $|o_i| = o_i$ *if* o_i *is a term-type variable.*
- $|\mathcal{T}_1 \to \mathcal{T}_2| = |\mathcal{T}_1| \to |\mathcal{T}_2|$
- $|(\mathcal{T}_1 \to \mathcal{S}) \Rightarrow \mathcal{T}_2| = |\mathcal{T}_1| \to |\mathcal{S} \Rightarrow \mathcal{T}_2|$
- $|\sigma_i \Rightarrow \mathcal{T}| = \sigma_i \to |\mathcal{T}|$
- $|\perp \Rightarrow \mathcal{T}| = \sigma_\perp \to |\mathcal{T}|$

This translation defines actually a translation of $\Lambda_{\mathcal{S}}$ types into simple types since all the pre-types of the same type are sent to the same simple type as is easily checked. We now translate typed $\Lambda\mu$-terms into λ-terms:

Definition 15. *For each stream variable α, one consider new variables of λ-calculus: $\alpha_0, \alpha_1, \ldots, \alpha_n, \ldots$ The translation is then defined as follows:*

$$\begin{aligned}
\lceil x \rceil^{\Lambda_{\mathcal{S}}} &= x \\
\lceil \lambda x^A.t \rceil^{\Lambda_{\mathcal{S}}} &= \lambda x^{|A|}.\lceil t \rceil^{\Lambda_{\mathcal{S}}} \\
\lceil (t)u \rceil^{\Lambda_{\mathcal{S}}} &= (\lceil t \rceil^{\Lambda_{\mathcal{S}}})\lceil u \rceil^{\Lambda_{\mathcal{S}}} \\
\lceil \mu\alpha^{A_1 \to \ldots A_n \to \sigma}.t \rceil^{\Lambda_{\mathcal{S}}} &= \lambda\alpha_1{}^{|A_1|}. \ldots \lambda\alpha_n{}^{|A_n|}.\lambda\alpha_{n+1}{}^{|\sigma|}.\lceil t \rceil^{\Lambda_{\mathcal{S}}} \\
\lceil (t)\alpha \rceil^{\Lambda_{\mathcal{S}}} &= (\lceil t \rceil^{\Lambda_{\mathcal{S}}})\alpha_1 \ldots \alpha_{n+1} \text{ if } \alpha \text{ is of type } A_1 \to \ldots A_n \to \sigma
\end{aligned}$$

where σ is either a stream-type variable or $\perp$.

Proposition 11. *If t is a typed $\Lambda\mu$-term, then $\lceil t \rceil^{\Lambda_{\mathcal{S}}}$ is a simply typed λ-term.*

Proposition 12 (Simulation). *Given two typed $\Lambda\mu$-terms t and u, the following facts hold:*

- *If $t \to_{fst^\to} u$ then $\lceil t \rceil^{\Lambda_{\mathcal{S}}} = \lceil u \rceil^{\Lambda_{\mathcal{S}}}$*
- *If $t \to_{\boldsymbol{\beta\eta}} u$ then $\lceil t \rceil^{\Lambda_{\mathcal{S}}} \to^+_{\beta\eta} \lceil u \rceil^{\Lambda_{\mathcal{S}}}$*

Proposition 13 ($fst^\to$ terminates). *Let t be a typed $\Lambda\mu$-term. there is no infinitely long $fst^\to$-derivation from t.*

Proposition 14 ($\lceil \_ \rceil^{\Lambda_{\mathcal{S}}}$ is reduction-length increasing). *If $t \to^\star_{\Lambda\mu^\to} u$ with m $\boldsymbol{\beta\eta}$-reduction steps, then there exists a $\beta\eta$-reduction from $\lceil t \rceil^{\Lambda_{\mathcal{S}}}$ to $\lceil u \rceil^{\Lambda_{\mathcal{S}}}$ in at least m reduction steps.*

We can finally prove strong normalization of typed $\Lambda\mu$-calculus:

Proof. Let us suppose that there exists an infinitely long typed reduction sequence starting at a typed $\Lambda\mu$-term t: $(t_i)_{i\geq 0}$ with $t = t_0$ and $t_i \to_{\Lambda\mu^\to} t_{i+1}$.

This reduction sequence contains only a finite number of $\boldsymbol{\beta\eta}$-reduction steps by proposition 14: otherwise we would obtain an infinite $\beta\eta$-reduction sequence from $\lceil t \rceil^{\Lambda_{\mathcal{S}}}$ in simply typed λ-calculus. Thus there is a integer n_0 such that for all $n \geq n_0$, $t_n \to_{fst^\to} t_{n+1}$, and thus we would have an infinitely long $fst^\to$ reduction sequence which contradicts termination of $fst^\to$. □

6 Conclusion

The aim of this paper was two-fold: to develop the meta-theory of $\Lambda\mu$-calculus, an extension of Parigot's $\lambda\mu$-calculus that we introduced to investigate Böhm theorem [18] and to review and compare different versions of call-by-name $\lambda\mu$-calculus that are found in the literature. Indeed, the relationships between those calculi was seldom made clear.

Contributions of the paper. The contributions of the paper are as follows:

- We proved confluence of $\Lambda\mu$-calculus and obtained a proof of the confluence for $\lambda\mu\eta$ which is simpler than the proof previously known from [17];
- We introduced a type system, $\Lambda_{\mathcal{S}}$, for $\Lambda\mu$-calculus that has subject reduction and strong normalization. $\Lambda_{\mathcal{S}}$ allows more terms to be typed and in particular terms that were need to obtain Böhm theorem in [18] whereas they were not typable in the usual type system for $\lambda\mu$-calculi.
- We investigated some relationships between call-by-name $\lambda\mu$-calculi. In particular, we proved that $\Lambda\mu$-calculus is a conservative extension of $\lambda\mu\eta$and that $\lambda\mu\epsilon$-calculus is a conservative extension of $\lambda\mu$.
- On the other hand, the equational theory of $\Lambda\mu$ and $\lambda\mu\epsilon$cannot be compared: for any $t \in \Sigma_{\Lambda\mu}$, there exist $u, v \in \Sigma_{\Lambda\mu}$ such that $t =_{\Lambda\mu} u$ and $t \neq_{\lambda\mu\epsilon} u$ and $t =_{\lambda\mu\epsilon} v$ and $t \neq_{\Lambda\mu} v$.
- The difference between $\Lambda\mu$ and $\lambda\mu\epsilon$ is also emphasized by the fact that $\Lambda\mu$ has Böhm theorem while in $\lambda\mu\epsilon$ it is not possible to separate $\mu\alpha.0$ and $\mu\alpha.1$.

Future works. We plan to develop this work in several directions:

- The comparison between $\Lambda\mu$ and $\lambda\mu\epsilon$ could probably be made more precise thanks to $\lambda\mu\widehat{\mathsf{tp}}$ that we are currently investigating with Herbelin and Ghilezan. This should in particular allow to understand more precisely where are the different limits to the expressiveness of the variants of $\lambda\mu$-calculus;
- The logical content of $\Lambda_{\mathcal{S}}$ is still to be made clearer: some results have been obtained in a joined work with Michele Pagani by connecting $\Lambda\mu$ to a sort of polarized proof-nets thanks to $\Lambda_{\mathcal{S}}$.
- The interpretation of $\Lambda\mu$-calculus as a stream calculus was essential in designing $\Lambda_{\mathcal{S}}$. We wish to develop this aspect by studying the relationships between $\Lambda\mu$ and infinitary λ-calculi.
- We wish to extend $\Lambda_{\mathcal{S}}$ in particular in the direction of polymorphism.
- Finally, an important motivation for providing $\Lambda\mu$ with a different type system than the original Parigot type system was to investigate typed separation. Indeed, there is no hope to obtain a typed separation result with a classical typing of $\Lambda\mu$ so that another type system, allowing more terms to be typed, was needed.

Acknowledgments: The author wishes to thank Michele Pagani, Hugo Herbelin and Silvia Ghilezan for helpful discussions and fruitful comments on a previous version of a part of this work [19].

References

1. Zena M. Ariola, Hugo Herbelin, and Amr Sabry. A type-theoretic foundation of delimited continuations. *Higher-order symbolic computation*, 2007.
2. Corrado Böhm. Alcune proprietà delle forme $\beta\eta$-normali nel λK-calcolo. *Publicazioni dell'Istituto per le Applicazioni del Calcolo*, 696, 1968.
3. René David and Walter Py. $\lambda\mu$-calculus and Böhm's theorem. *Journal of Symbolic Logic*, 2001.
4. Philippe de Groote. On the relation between the $\lambda\mu$-calculus and the syntactic theory of sequential control. In *LPAR'94*, volume 822 of *LNAI*, 1994.
5. Philippe de Groote. An environment machine for the $\lambda\mu$-calculus. *MSCS*, 8, 1998.
6. Kosta Dosen and Zoran Petric. The typed Böhm theorem. *ENTCS*, 50(2), 2001. in Proceedings of BOTH 2001.
7. Timothy Griffin. A formulae-as-types notion of control. In *POPL'90*, 1990.
8. Hugo Herbelin and Silvia Ghilezan. An approach to call-by-name delimited continuations. In *POPL*, january 2008.
9. William A. Howard. The formulae-as-type notion of construction, 1969. In J. P. Seldin and R. Hindley, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, New York, 1980.
10. Thierry Joly. *Codages, séparabilité et représentation de fonctions en λ-calcul simplement typé et dans d'autres systèmes de types.* PhD thesis, Université Paris VII, 2000.
11. Luke Ong. A semantic view of classical proofs. In *LICS'96*, 1996.
12. Michele Pagani and Alexis Saurin. Stream Associative Nets and Lambda-mu-calculus. Technical Report RR-6431, INRIA, January 2008.
13. Michel Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In *Proceedings of the 1992 International Conference on Logic Programming*, volume 624 of *LNCS*, London, UK, 1992. Springer-Verlag.
14. Michel Parigot. Classical proofs as programs. In *Proceedings of the Third Kurt Gödel Colloquium on Computational Logic and Proof Theory*, volume 713 of *LNCS*, pages 263 – 276, London, UK, 1993. Springer-Verlag.
15. Michel Parigot. Strong normalization for second order classical natural deduction. In M. Vardi, editor, *Eighth Annual Symposium on Logic in Computer Science*, pages 39–46. IEEE, June 1993.
16. Michel Parigot. Proofs of strong normalisation for second order classical natural deduction. *Journal of Symbolic Logic*, 62(4):1461–1479, december 1997.
17. Walter Py. *Confluence en $\lambda\mu$-calcul.* PhD thesis, Université de Savoie, 1998.
18. Alexis Saurin. Separation with streams in the $\Lambda\mu$-calculus. In *Twentieth Annual Symposium on Logic in Computer Science.* IEEE, 2005.
19. Alexis Saurin. Typing streams in the $\Lambda\mu$-calculus. In *Short paper presented at LPAR'07*, 2007.
20. Richard Statman. λ-definable functionals and $\beta\eta$ conversion. *Archiv für Mathematische Logik und Grundlagenforschung*, 22:1–6, 1983.