Typing Streams in the $\Lambda\mu$-Calculus

ALEXIS SAURIN
INRIA Saclay and École Polytechnique

$\Lambda\mu$-calculus is a Böhm-complete extension of Parigot's $\lambda\mu$-calculus closely related with delimited control in functional programming. In this article, we investigate the meta-theory of untyped $\Lambda\mu$-calculus by proving confluence of the calculus and characterizing the basic observables for the Separation theorem, *canonical normal forms*. Then, we define $\Lambda_{\mathcal{S}}$, a new type system for $\Lambda\mu$-calculus that contains a special type construction for streams, and prove that strong normalization and type preservation hold. Thanks to the new typing discipline of $\Lambda_{\mathcal{S}}$, new computational behaviors can be observed, which were forbidden in previous type systems for $\lambda\mu$-calculi. Those new typed computational behaviors witness the stream interpretation of $\Lambda\mu$-calculus.

Categories and Subject Descriptors: D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*Syntax*; D.3.2 [**Programming Languages**]: Language Classifications—*Applicative (functional) languages*; D.3.3 [**Programming Languages**]: Language Constructs and Features—*Control structures data types and structures*; F.3.3 [**Logics and Meanings of Programs**]: Studies of Program Constructs—*Control primitives; functional constructs; type structure*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Lambda calculus and related systems; proof theory*

General Terms: Languages, Theory

Additional Key Words and Phrases: Böhm theorem, classical λ-calculus, confluence, delimited control, $\lambda\mu$-calculus, streams, type system

ACM Reference Format:

Saurin, A. 2010. Typing streams in the $\Lambda\mu$-calculus. ACM Trans. Comput. Logic, 11, 4, Article 28 (July 2010), 34 pages.
DOI = 10.1145/1805950.1805958 http://doi.acm.org/10.1145/1805950.1805958

1. INTRODUCTION

Curry-Howard in Classical Logic. Curry-Howard correspondence [Howard 1980] was first designed as a correspondence between minimal logic and simply typed λ-calculus. Extending the correspondence to classical logic took more than two decades but it resulted in strong connections with control operators in functional programming languages as first noticed by Griffin [1990] who

Author's address: LIX, École Polytechnique, Route de Saclay, 91128 Palaiseau Cedex, France; email: alexis.saurin@inria.fr.

DOI 10.1145/1805950.1805958 http://doi.acm.org/10.1145/1805950.1805958

analyzed the logical content of Felleisen's $\mathcal{C}$ operator [Felleisen et al. 1987] by observing that it can be given the type of double-negation elimination. Shortly after Griffin, several other proposals have been made for a computational interpretation of classical logic (Parigot's $\lambda\mu$-calculus [Parigot 1992], Girard's LC [Girard 1991; Murthy 1992], ...) deepening the logical analysis started by Griffin. In particular, Parigot introduced $\lambda\mu$-calculus [Parigot 1992] as an extension of λ-calculus isomorphic to an alternative presentation of classical natural deduction [Parigot 1991] in which one can encode usual control operators (`call/cc`, $\mathcal{C}$ and $\mathcal{A}$ operators).

$\lambda\mu$-Calculus and Separation. $\lambda\mu$-calculus became one of the most widely studied classical lambda-calculi, both in the typed and untyped settings for several reasons: its simplicity, the fact that it extends intuitionistic natural deduction in a straightforward way and the fact that it naturally extends λ-calculus while retaining most of λ-calculus standard properties, contrarily to some other proposals. However, a fundamental property of pure λ-calculus, known as separation property (or Böhm theorem [Böhm 1968]), does not hold for $\lambda\mu$-calculus as shown by David and Py [2001]. In a previous work, we introduced an extension to $\lambda\mu$-calculus, $\Lambda\mu$-calculus, for which we proved that separation holds [Saurin 2005] and which admits a stream interpretation: $\Lambda\mu$-calculus is a Böhm-complete extension of Parigot's $\Lambda\mu$-calculus. Interestingly, the fact that $\Lambda\mu$-calculus validates Böhm theorem seems to be closely related to its ability to capture delimited control which is a very expressive class of control operators.

The first aim of this article is to study the metatheory of $\Lambda\mu$-calculus.

(i) We prove a confluence theorem for $\Lambda\mu$-calculus.
(ii) We then study and characterize $\Lambda\mu$-calculus canonical normal forms that are the values, or observables, for the separation results in $\Lambda\mu$-calculus.
(iii) We shall compare different variants of $\lambda\mu$-calculi.

Variants of $\lambda\mu$-Calculi. Indeed, several variants of Parigot's $\lambda\mu$-calculus are often considered in the literature. Their relationship is usually not made clear: semantics of those languages can differ as well as their syntactical properties. For instance, $\Lambda\mu$-calculus satisfies Böhm theorem while original Parigot's calculus does not. Moreover, it is sometimes unclear what properties are true in which variant of $\lambda\mu$. One of the purposes of this article is thus to study and compare different variants of call-by-name $\lambda\mu$-calculus in order to stress their differences as well as their relationships and to understand how it may affect the expressiveness and semantics of those calculi.

Typed Separation. In addition to Böhm theorem which holds for pure λ-calculus, several authors [Statman 1983; Joly 2000; Došen and Petrić 2000, 2001] have evidenced a property of the same kind in simply-typed λ-calculus. Even though they are slightly more restricted than Böhm theorem for the untyped λ-calculus, those typed separation results are still very interesting: they show the maximality of $\beta\eta$-equivalence in typed λ-calculus and, by the

Curry-Howard correspondence, they provide intuitionistic natural deduction with such a separation result.

Those typed Böhm theorems for simply-typed λ-calculus are our initial motivation for a deeper study of $\Lambda\mu$-calculus type system. This leads us to the second main contribution of this paper: the introduction of a new type system for $\Lambda\mu$-calculus, Λ_S, for which we prove that standard properties hold (type preservation, strong normalization, ...) and that we regard as a first step towards a typed separation theorem for $\Lambda\mu$-calculus (that we leave for future work) and towards a logical interpretation of delimited control.

Structure of the Article. In Section 2, we first recall Parigot's $\lambda\mu$-calculus as well as David and Py's result about the failure of separation for $\lambda\mu$ which allows us to introduce a Böhm-complete extension of Parigot's calculus, $\Lambda\mu$-calculus; we recall $\Lambda\mu$-calculus' intuitive stream interpretation that shall be important when designing Λ_S type system. For sake of completeness we finish this section by considering several other variants of Parigot's $\lambda\mu$-calculus considered in the literature. Second, we develop the meta-theory of pure $\Lambda\mu$-calculus in Section 3 by proving a confluence theorem, characterizing the $\Lambda\mu$-canonical normal forms and comparing the equational theories of variants of $\lambda\mu$-calculi. Finally, we introduce in Section 4 a new type system for $\Lambda\mu$-calculus, Λ_S, which differs from the usual classical type system for $\lambda\mu$-calculus but retains the main properties of interest and allows, in the typed setting, to observe the main computational features that make $\Lambda\mu$-calculus differ from $\lambda\mu$-calculus.

This article extends the conference paper *On the relations between the syntactic theories of $\lambda\mu$-calculi* [Saurin 2008a] with results from the author's doctoral dissertation entitled *Une étude logique du contrôle* [Saurin 2008b].

Notations and Conventions. We shall use Krivine's notation [Krivine 1993] for λ-terms (resp. $\lambda\mu$, $\Lambda\mu$-terms): λ-application shall be written $(t)u$ and λ-application is be left-associative, that is $(t)u_1 \cdots u_{k-1}u_k$ shall be read as $(\cdots((t)u_1)\cdots u_{k-1})u_k$. Moreover we shall use in this paper an alternative notation for $\lambda\mu$-terms that we introduced and justified in a previous work [Saurin 2005, 2008b], writing $(t)\alpha$ instead of the more common $[\alpha]t$. Terms shall always be considered up to α-equivalence.

2. FROM $\lambda\mu$ TO $\Lambda\mu$

The aim of this section is to introduce $\Lambda\mu$-calculus by reviewing different variants of $\lambda\mu$-calculi from the point of view of the separation property: starting with Parigot's original presentation of $\lambda\mu$-calculus, we discuss the failure of separation in Parigot's calculus and define $\Lambda\mu$-calculus as a natural extension for which separation property holds. Last, we recall several other variants of call-by-name $\lambda\mu$-calculus often considered in the literature.

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" [Parigot 1992] by

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

$$\frac{\Gamma, x : A \vdash_{\lambda\mu} t \ : B|\Delta}{\Gamma \vdash_{\lambda\mu} \lambda x^A.t \ : A \to B|\Delta}\ \lambda\text{-Abs} \qquad \frac{\Gamma \vdash_{\lambda\mu} t \ : A \to B|\Delta \qquad \Gamma \vdash_{\lambda\mu} u \ : A|\Delta}{\Gamma \vdash_{\lambda\mu} (t)u \ : B|\Delta}\ \lambda\text{-App}$$

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

allowing for a proof-program correspondence *à la* Curry-Howard [Howard 1980] between $\lambda\mu$-calculus and classical natural deduction [Parigot 1991, 1992]. $\lambda\mu$-calculus extends Church's λ-calculus with a second abstraction, μ-abstraction.

Definition 2.1 ($\Sigma_{\lambda\mu}$). Terms of Parigot's $\lambda\mu$-calculus (or *$\lambda\mu$-terms*: $t, u, v, \cdots \in \Sigma_{\lambda\mu}$) are defined by the following syntax:

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

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 closed $\lambda\mu$-terms is written $\Sigma^c_{\lambda\mu}$.

In $\mu\alpha.(t)\beta$, variable β is in the scope of $\mu\alpha$. Terms of the form $(t)\alpha$ are not elements of $\Sigma_{\lambda\mu}$, but are usually referred to as *named terms* and are generically written n.

Definition 2.2 (*$\lambda\mu$-Calculus Reduction*). *$\lambda\mu$-calculus reduction*, written $\longrightarrow_{\lambda\mu}$, is the reduction induced by the following four reduction rules:

$$\boxed{\begin{array}{llll} (\lambda x.t)u & \longrightarrow_\beta & t\{u/x\} & \\ (\mu\alpha.n)u & \longrightarrow_\mu & \mu\alpha.n\{(v)u\alpha/(v)\alpha\} & \\ (\mu\alpha.n)\beta & \longrightarrow_\rho & n\{\beta/\alpha\} & \\ \mu\alpha.(t)\alpha & \longrightarrow_\theta & t & \text{if } \alpha \notin FV(t) \end{array}}$$

$n\{(v)u\alpha/(v)\alpha\}$ substitutes (without variable-capture) every subterms of the form $(v)\alpha$ in n by $((v)u)\alpha$ as shown by the following definition:

Definition 2.3 (*Structural Substitution*). *Structural substitution* is defined on $\Sigma_{\lambda\mu}$ and on named terms, as follows:

$$\begin{array}{llll} x & \{(v)u\alpha/(v)\alpha\} = & x & \\ \lambda x.t & \{(v)u\alpha/(v)\alpha\} = & \lambda x.t\{(v)u\alpha/(v)\alpha\} & (\text{if } x \notin FV(u)) \\ (t)t' & \{(v)u\alpha/(v)\alpha\} = & (t\{(v)u\alpha/(v)\alpha\})t'\{(v)u\alpha/(v)\alpha\} & \\ \mu\alpha.n & \{(v)u\alpha/(v)\alpha\} = & \mu\alpha.n & \\ \mu\beta.n & \{(v)u\alpha/(v)\alpha\} = & \mu\beta.n\{(v)u\alpha/(v)\alpha\} & (\text{if } \beta \neq \alpha \text{ and } \beta \notin FV(v)) \\ (t)\alpha & \{(v)u\alpha/(v)\alpha\} = & (t\{(v)u\alpha/(v)\alpha\})v\alpha & \\ (t)\beta & \{(v)u\alpha/(v)\alpha\} = & (t\{(v)u\alpha/(v)\alpha\})\beta & \end{array}$$

Parigot's $\lambda\mu$-calculus thus extends λ-calculus by providing a Curry-Howard correspondence with a natural-deduction-like presentation of classical logic thanks to the typing system presented in Figure 1. Moreover, $\lambda\mu$ satisfies lots of sandard properties of λ-calculus, in particular confluence of the untyped calculus [Parigot 1992; Py 1998; David and Py 2001; Baba et al. 2001], subject

reduction [Parigot 1992] and strong normalization [Parigot 1993, 1997] of the typed calculus (see Figure 1).

2.2 $\lambda\mu$-Calculus with Extensionality: Py's $\lambda\mu\eta$

There is quite a long story about confluence property in $\lambda\mu$-calculus. Indeed, the original proof published by Parigot [1992] contained a flaw: the parallel reduction method (also known as Tait-Martin-Löf method) used in that paper was not powerful enough to ensure that the parallel reduction was strongly confluent (for it did not treat a case where a ρ-redex is "blocked" by a μ-redex) and thus that the original reduction was confluent. Py proposed in his PhD dissertation [Py 1998; David and Py 2001] a corrected version of the proof that was later, but independently, discovered by Baba et al. [2001] following the same method in a slightly restricted case: indeed, Baba et al. [2001] do not consider Parigot's θ-reduction.

Moreover, Parigot [1992] only considered reductions β, μ, ρ and θ and did not introduce rule η, from λ-calculus, for extensionality. Indeed, whereas the critical pair between β and η does not prevent Church-Rosser to hold, the critical pair μ/η does not converge:

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

Indeed, take for instance n to be $(\lambda y.\mu\beta.(y)\alpha)\alpha$ in the previous example, then the resulting term has two distinct normal forms: $\mu\alpha.(\lambda y.\mu\beta.(y)\alpha)\alpha$ and $\lambda x.(x)x$. This witness that confluence does not hold. The nonconverging μ/η critical pair is probably the reason why η is not considered in Parigot's original presentation.

In his PhD dissertation, Py studied confluence properties of $\lambda\mu$-calculus and was interested in Böhm theorem for $\lambda\mu$-calculus. He thus had to consider seriously the question of extensionality which resulted in the addition of η to $\lambda\mu$-calculus, confluence being restored with an additional rule, ν, that makes the above diagram converge.[1]

$\lambda\mu\eta$-calculus is the calculus on $\Sigma_{\lambda\mu}$ obtained by adding the following rules to the reductions of $\lambda\mu$-calculus:

$$\begin{array}{lll} \lambda x.(t)x \longrightarrow_{\eta} t & & \text{if } x \notin FV(t) \\ \mu\alpha.n \longrightarrow_{\nu} \lambda x.\mu\alpha.n\{(v)x\alpha/(v)\alpha\} & & \text{if } x \notin FV(n) \end{array}$$

2.3 Failure of Separation Property in $\lambda\mu\eta$-Calculus

David and Py proved that separation fails in $\lambda\mu\eta$ [Py 1998; David and Py 2001] by finding a counter-example to separation. They exhibit two terms $w_0, w_1 \in \Sigma_{\lambda\mu}$ (obtained by substituting y by $0 = \lambda x, x'.x'$ and $1 = \lambda x, x'.x$ respectively in $w_y = \lambda x.\mu\alpha.((x)\mu\beta.(x)u_0y\alpha)u_0\alpha$ with $u_0 = \mu\delta.(\lambda z_1, z_2.z_2)\alpha$) that are solvable, not equivalent for the equivalence relation induced by $\lambda\mu\eta$-reduction rules but that no context can distinguish. The nonseparation behavior is shown in Figure 2 and described in details by David and Py [2001] and Saurin [2005, 2008b]. In order to achieve a Böhm out in an applicative context $[]\vec{AB}$, the first occurrence

[1]Reduction ν was already briefly considered by Parigot in his seminal article [Parigot 1992].

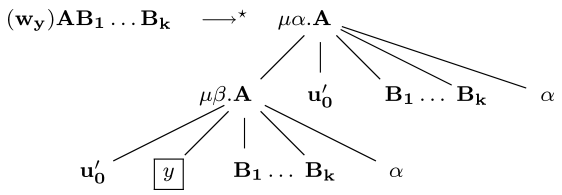


Fig. 2. Counter-example to separation in $\lambda\mu\eta$.

of A shall select its first argument while the second occurrence of A shall select its second argument. However, those two different computational behaviors shall be realized in the same evaluation context $[]B_1 \cdots B_k\alpha$ which is impossible by confluence.

2.4 A $\lambda\mu$-Calculus Satisfying Böhm Theorem: $\Lambda\mu$-Calculus

The failure of separation in $\lambda\mu\eta$-calculus may be understood as the fact that some separating contexts are missing in $\lambda\mu\eta$ for separation to hold, resulting in the impossibility to observe y in the example of Figure 2. This led us to define, in a previous work [Saurin 2005], $\Lambda\mu$-calculus, an extension to $\lambda\mu\eta$ for which we proved Böhm theorem. In $\Lambda\mu$-calculus, the validity of separation may be understood as the fact that the new contexts made available by the new syntax are sufficiently discriminating to realize a Böhm Out (in the new syntax).

Definition 2.4 ($\Sigma_{\Lambda\mu}$). $\Lambda\mu$-terms are defined by the following syntax:

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

where x ranges over a set $\mathcal{V}_t$ of term variables and α ranges over a set $\mathcal{V}_s$ of stream variables. $\mathcal{V}_t$ and $\mathcal{V}_s$ are disjoint. Closed $\Lambda\mu$-terms are denoted by $\Sigma^c_{\Lambda\mu}$.

It is clear, since $\alpha \notin \Sigma_{\Lambda\mu}$, that there is no ambiguity between notation $(t)\alpha$ and $(t)u$. Notice that $\Sigma_{\lambda\mu} \subsetneq \Sigma_{\Lambda\mu}$ since named terms of definition 2.1 are elements of $\Sigma_{\Lambda\mu}$ but not of $\Sigma_{\lambda\mu}$. Moreover, terms such as $\mu\alpha.\mu\beta.t$ or $\lambda x.(t)\alpha y$ are elements of $\Sigma_{\Lambda\mu} \setminus \Sigma_{\lambda\mu}$.

Definition 2.5 ($\Lambda\mu$-*Calculus Reduction*). $\Lambda\mu$-calculus reduction, written $\longrightarrow_{\Lambda\mu}$, is induced by the following five reduction rules:

$(\lambda x.t)u$	$\longrightarrow_{\beta_T}$	$t\{u/x\}$	
$\lambda x.(t)x$	$\longrightarrow_{\eta_T}$	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 $x \notin FV(t)$

Remark 2.6. β_T, η_T, β_S, η_S, *fst* correspond respectively to $\beta, \eta, \rho, \theta$ and ν; see [Saurin 2005, 2008b] for details. Notice that μ is not part of $\Lambda\mu$-calculus reduction system but can be simulated by a *fst*-reduction followed by a β_T-reduction: $(\mu\alpha.t)u \longrightarrow_{fst} (\lambda x.\mu\alpha.t\{(v)x\alpha/(v)\alpha\})u \longrightarrow_{\beta_T} \mu\alpha.t\{(v)u\alpha/(v)\alpha\}$.

Names for reduction rules in $\Lambda\mu$-calculus come from the stream interpretation of this calculus: $\mu\alpha$ can be considered as an abstraction over streams of terms while $(t)\alpha$ is seen as the construction feeding a program t with a stream of inputs. Thus, μ is a sort of infinitary λ-abstraction but in a much stronger sense than what Parigot remarked in his seminal article [Parigot 1992], see for instance [Saurin 2005, 2008b] for details as well as the part on typing in this article. β_T and η_T are the usual reductions of λ-calculus; β_S and η_S are respectively a reduction for stream β-reduction and an extensionality rule on streams while *fst*-reduction relates both constructions: it instantiates the first elements of a stream:

$$\boxed{\mu\alpha.t \longrightarrow^{\star}_{fst} \lambda x_1 \cdots \lambda x_n.\mu\alpha.t\{(v)x_1 \ldots x_n\alpha/(v)\alpha\}}$$

A separation result is stated with respect to a set of observables (in λ-calculus, they are the $\beta\eta$-normal forms). Since *fst* is an expansion rule, there are very few normal forms in $\Lambda\mu$. We thus consider a set of canonical normal forms [David and Py 2001; Saurin 2005, 2008b] as basic observables for separation:

Definition 2.7. A $\Lambda\mu$-term t is in *canonical normal form (CNF)* if it is $\beta_T\eta_T\beta_S\eta_S$-normal and if it contains no subterm of the form $(\lambda x.u)\alpha$ nor $(\mu\alpha.u)v$.

Remark 2.8. A closed canonical normal form is thus a $\beta_T\eta_T\beta_S\eta_S$-normal form such that no *fst*-reduction creates a β_T-redex.

Definition 2.9. $\Lambda\mu$-contexts are defined by the following grammar:

$$\boxed{\mathcal{C} ::= [] \mid \lambda x.\mathcal{C} \mid (t)\mathcal{C} \mid (\mathcal{C})t \mid \mu\alpha.\mathcal{C} \mid (\mathcal{C})\alpha.}$$

A context is said to be *stream applicative* if it follows the following grammar:

$$\boxed{\mathcal{C} ::= [] \mid (\mathcal{C})t \mid (\mathcal{C})\alpha.}$$

that is if it is of the form

$$\boxed{[]t_1^1 \ldots t_{n_1}^1\alpha_1 \ldots t_1^k \ldots t_{n_k}^k\alpha_k t_1^{k+1} \ldots t_{n_{k+1}}^{k+1}.}$$

We can now state the separation result for $\Lambda\mu$-calculus [Saurin 2005, 2008b]:

THEOREM 2.10 (BÖHM THEOREM FOR $\Lambda\mu$-CALCULUS). *If t and t' are two closed canonical normal forms that are not $\Lambda\mu$-equivalent, then there exists a (stream applicative) context $\mathcal{C}[]$ such that:*

$$\begin{array}{lll} \mathcal{C}[t] & \longrightarrow^{\star}_{\Lambda\mu} & \lambda x.\lambda y.x \quad \textit{and} \\ \mathcal{C}[t'] & \longrightarrow^{\star}_{\Lambda\mu} & \lambda x.\lambda y.y. \end{array}$$

Remark 2.11. The structure of stream discussed in remark 2.6 is extensively used when proving Theorem 2.10 via the structure of *Parametric Pairs* which is crucial for our Böhm Out technique:

$$\boxed{\langle t,u\rangle_k = \mu\alpha_1 \ldots \mu\alpha_k.\lambda x.((x)(t)\alpha_1 \ldots \alpha_k)(u)\alpha_1 \ldots \alpha_k}$$

David Py's counter-example can be separated by the following stream applicative context:

$$\boxed{\mathcal{C} = ([])\mathcal{P}x_0x_1\alpha 0\alpha 1\alpha}$$

where $\mathcal{P}$ is the parametric pair: $\langle \lambda z_0, z_1.\mu\alpha.\mu\beta.z_1, \lambda x.\mu\alpha.x\rangle_1$.

The following notations for subsytems of $\Lambda\mu$-reduction will be useful in the rest of the article:

Definition 2.12 ($\boldsymbol{\beta}$, $\boldsymbol{\beta}^{var}$, $\boldsymbol{\eta}$, fst^-). We consider the following subsystems of $\Lambda\mu$-reduction or $\Lambda\mu$-equivalence.

(1) $\boldsymbol{\beta}$ is the subsystem made of reductions β_T and β_S.
(2) $\boldsymbol{\eta}$ is the subsystem made of reductions η_T and η_S.
(3) $\boldsymbol{\beta}_{fst}$ is the subsystem $\beta_T\beta_S fst$.
(4) $\boldsymbol{\beta\eta}fst$ stands for the full $\Lambda\mu$-reduction system.
(5) $\boldsymbol{\beta}^{var}$ for the subsystem of $\boldsymbol{\beta}$ that reduces a $\boldsymbol{\beta}$-redex only when the argument is a variable, for instance $(\lambda x.\mu\alpha.t)y\beta \longrightarrow^{\star}_{\boldsymbol{\beta}^{var}} t\{y/x\}\{\beta/\alpha\}$.
(6) $\boldsymbol{\beta}^{var}\boldsymbol{\eta}$, $\boldsymbol{\beta}^{var}{}_{fst}$ and $\boldsymbol{\beta}^{var}\boldsymbol{\eta}_{fst}$ are straightforward.
(7) fst^- is the restriction of fst to redexes $t = \mu\alpha.t'$ which are applied to a term u or such that t' contains at least one subterm $(\lambda x.u)\alpha$.
(8) $\longrightarrow_{\Lambda\mu^-}$ is $\longrightarrow_{\boldsymbol{\beta\eta}fst^-}$.
(9) $=_{\Lambda\mu}$ and $=_{\Lambda\mu^-}$ are the equivalence relations corresponding, respectively, to $\longrightarrow_{\Lambda\mu}$ and $\longrightarrow_{\Lambda\mu^-}$.

2.5 Other Variants of CBN $\lambda\mu$-Calculus on $\Sigma_{\Lambda\mu}$

de Groote [1994] introduced an extension to Parigot's $\lambda\mu$-calculus while comparing $\lambda\mu$-calculus with Felleisen's $\lambda\mathcal{C}$ [Felleisen et al. 1987; Felleisen and Hieb 1992] that we shall refer to as $\lambda\mu_{dG}$. $\lambda\mu_{dG}$-calculus is a close relative of $\Lambda\mu$-calculus but it has neither η_T nor fst, it can thus be regarded as a subcalculus of $\Lambda\mu$. Ong [1996] also considers a calculus built on $\Sigma_{\Lambda\mu}$ which is also very close to $\Lambda\mu$ but it is presented equationally.

We shall detail another variant of Parigot's $\lambda\mu$-calculus introduced by de Groote [1998] while studying an abstract machine for $\lambda\mu$-calculus. In order to do so, he 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[2]:

$$\boxed{\mu\alpha.\mu\beta.t \to_{\epsilon} \mu\alpha.|t|_{\beta}}$$

where $|t|_{\beta}$ is the result of removing all the free occurrences of β in t. A constraint on reduction ρ is added in order not to lose confluence: ρ-reduction is now $\mu\gamma.(\mu\alpha.t)\beta \longrightarrow_{\rho} \mu\gamma.t\{\beta/\alpha\}$ that is only $\Sigma_{\Lambda\mu}$-terms which satisfy locally $\Sigma_{\lambda\mu}$'s grammar can be subject to a ρ-reduction. Otherwise, there is a critical pair on

[2]Actually, reduction ϵ is already considered in de Groote [1994] while encoding the abort operator in $\lambda\mu$-calculus.

$(\mu\gamma.\mu\beta.\mu\alpha.t)\delta\zeta$ depending on whether we apply ϵ to $\mu\alpha$ or $\mu\beta$:

$$\begin{array}{rl}(\mu\gamma.\mu\beta.\mu\alpha.t)\delta\zeta & \longrightarrow_\epsilon (\mu\gamma.\mu\alpha.|t|_\beta)\delta\zeta \longrightarrow^2_\rho |t|_\beta\{\delta/\gamma\}\{\zeta/\alpha\} \\ & \longrightarrow_\epsilon (\mu\gamma.\mu\beta.|t|_\alpha)\delta\zeta \longrightarrow^2_\rho |t|_\alpha\{\delta/\gamma\}\{\zeta/\beta\}\end{array}$$

Moerover adding η-reduction to $\lambda\mu\epsilon$ results in the loss of confluence[3]:

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

Definition 2.13 ($\lambda\mu\epsilon$-*Calculus Reduction*). $\lambda\mu\epsilon$-calculus reduction, which is written $\longrightarrow_{\lambda\mu\epsilon}$, is induced by the following five reduction rules:

$$\begin{array}{lll}(\lambda x.t)u & \longrightarrow_\beta & t\{u/x\} \\ (\mu\alpha.t)u & \longrightarrow_\mu & \mu\alpha.t\{(v)u\alpha/(v)\alpha\} \\ \mu\gamma.(\mu\alpha.t)\beta & \longrightarrow_\rho & \mu\gamma.t\{\beta/\alpha\} \\ \mu\alpha.(t)\alpha & \longrightarrow_\theta & t \qquad \text{if } \alpha \notin FV(t) \\ \mu\alpha.\mu\beta.t & \longrightarrow_\epsilon & \mu\alpha.|t|_\beta\end{array}$$

3. META-THEORY OF UNTYPED $\Lambda\mu$-CALCULUS

The present section is devoted to results on the meta-theory of pure $\Lambda\mu$-calculus. We first prove confluence of $\Lambda\mu$-calculus that is an important step in justifying $\Lambda\mu$-calculus as a suitable calculus for the study of control operators and which is important for separation to make sense. Moreover, our proof provides, as a corollary, a proof of confluence for $\lambda\mu\eta$-calculus that is slightly simpler than Py's original proof. We then turn our attention to canonical normal forms and provide a precise characterization of these terms which are the basic observables for separation in $\Lambda\mu$-calculus. We finally compare $\lambda\mu$-calculus and $\Lambda\mu$-calculus equational theories.

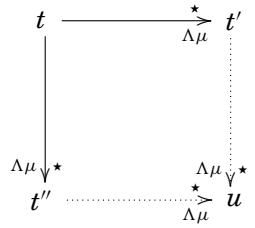
3.1 Confluence Theorem for $\Lambda\mu$-Calculus

$\Lambda\mu$-calculus originated in an analysis of the failure of separation in $\lambda\mu\eta$: $\Lambda\mu$ was designed in order to have separation. The question of confluence is thus a crucial one since a separation result for a calculus that does not have confluence is not very meaningful: one indeed needs a sort of property of uniqueness of normal forms (here canonical normal forms) or at least a simple criterion to check whether two (canonical) normal forms are equivalent or not. The confluence result proved in this section is thus important to enforce the separation result for $\Lambda\mu$-calculus [[Saurin 2005, 2008b]].

We aim at establishing the following confluence theorem for $\Lambda\mu$-calculus:

[3]Many details on problems of confluence in $\lambda\mu\epsilon$ with η can be found in Py's thesis [Py 1998].

THEOREM 3.1. *$\Lambda\mu$-calculus is confluent on μ-closed terms. For any $t, t', t'' \in \Sigma_{\Lambda\mu}$, μ-closed, there exists $u \in \Sigma_{\Lambda\mu}$ such that:*



As previously noticed, confluence in $\lambda\mu$ has quite a long story. Py [1998] proved confluence for $\lambda\mu\eta$-calculus, but this does not entail confluence of $\Lambda\mu$-calculus which strictly contains Py's $\lambda\mu\eta$-calculus. On the other hand, we shall see that confluence in $\lambda\mu\eta$-calculus is an immediate corollary of confluence for $\Lambda\mu$-calculus. In the following, we shall be inspired by Py's proof technique but, as we shall detail in the end of this section, our proof actually allows to simplify some arguments of the confluence proof for $\lambda\mu\eta$-calculus.

3.2 μ-Closure Requirement

We do not prove confluence for every terms in $\Sigma_{\Lambda\mu}$ but only for those which have no free stream-variable (the μ-closed terms). This restriction is already present when working with Parigot's syntax, $\Sigma_{\lambda\mu}$, see Py [1998]. We shall explain this restriction in the present paragraph.

Confluence problems in $\Lambda\mu$-calculus come from reductions involving both term variables and stream variables, that is both λ-abstractions and μ-abstractions. The two kinds of $\Lambda\mu$-variables are connected thanks to *fst*-rule and the high degree of complexity of the problem of confluence in $\Lambda\mu$-calculus comes from this reduction.

Reduction *fst* (or reduction ν using Parigot and Py's vocabulary) was introduced in $\lambda\mu$-calculus in order to solve a critical pair involving μ and η. But still, it is important that the stream variables are bound; indeed, there is a critical pair between reduction *fst* and reduction β_S:

$$\begin{array}{ccc} \mathcal{C}[(\mu\alpha.t)\beta] & \xrightarrow[fst]{} & \mathcal{C}[(\lambda x.\mu\gamma.t\{(u)x\gamma/(u)\alpha\})\beta] \\ \Big\downarrow{\scriptstyle \beta_S} & & \\ \mathcal{C}[t\{\beta/\alpha\}] & & \end{array}$$

that requires variable β to be bound in order for the confluence diagram to converge: for this critical pair to converge, one shall create a β_T-redex involving the λx abstraction in order to create a β_S-redex at $(\mu\gamma.t')\beta'$ at the next step, while a *fst*-reduction is applied to β in $t\{\beta/\alpha\}$. This is possible only if variable β is bound in context $\mathcal{C}$.

Another, simpler example is $t = (\mu\beta.x)\alpha$ which may reduce to $t' = x$, which is a $\Lambda\mu$-normal form, thanks to a β_S-reduction, or to $t'' = (\lambda y.\mu\gamma.x)\alpha$ by a *fst*-reduction. t'' cannot reduce to x anymore (there is only one reduction sequence from t'': $t'' \longrightarrow^{\star}_{fst} (\lambda y_1 \ldots y_n.\mu\gamma.x)\alpha$).

We thus have no hope to prove confluence without the μ-closure requirement, which is actually already required by Py [1998] for confluence in $\lambda\mu\eta$-calculus. In the following, terms are always considered to be μ-closed. For simplicity, we may sometimes consider $t, t', t'' \in \Sigma^c_{\Lambda\mu}$ in the following, but having free term variables does not change anything in the results which remain true (with no change to be made to the proofs) if there are free term variables.

Remark 3.2. Notice, however, that in a joint work with Pagani [Pagani and Saurin 2008; Saurin 2008b], we analyze $\Lambda\mu$-calculus through a version of Girard's proof nets providing a purely local account of the reduction rules (in particular of the *fst*-reduction) and thus obtaining confluence in proof nets without the hypothesis of μ-closure. This is due to the highly local reduction in the nets resulting in the possibility to initiate the reduction corresponding to a *fst*-reduction both at the μ-abstraction level but also at every subnet corresponding to a construction of the form $(t)\alpha$.

3.3 Proof of Theorem 3.1

The proof of Theorem 3.1 relies on several lemmas. The proof of the theorem is as follows:

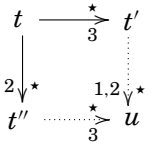
PROOF. Confluence of $\Lambda\mu$-calculus follows from confluence of $\boldsymbol{\beta}$*fst* (Proposition 3.22), confluence of $\boldsymbol{\eta}$ (Proposition 3.23) and commutation of the two previous systems (Proposition 3.24): thanks to Hindley-Rosen lemma, this ensures confluence of $\Lambda\mu$-calculus. □

Recall that Hindley-Rosen lemma ensures that the union of two confluent rewriting systems is confluent as soon as the rewriting systems commute. The following two commutation results, which are standard, will be useful in the rest of this section:

LEMMA 3.3. *Given three reductions* $\to_1$, $\to_2$ *and* $\to_3$, *if both* $\to_1$ *and* $\to_2$ *commute with* $\to_3$, *then* $\to_{1,2}$ *commutes with* $\to_3$.

The previous lemma is referred to as the commutation of the union of two reductions while the following lemma is a simple variant of the previous one:

LEMMA 3.4. *If* $\to_1$, $\to_2$ *et* $\to_3$ *are such that* $\to_1$ *commutes with* $\to_3$ *and that*



then $\to_{1,2}$ *commutes with* $\to_3$.

Remark 3.5. The difficult part of the proof will consist in proving confluence of $\boldsymbol{\beta}$*fst* to which will be devoted the next two subsections: we shall first prove several commutation and confluence properties of reduction systems made of β_T, β_S and *fst* before introducing an indexed variant of $\Lambda\mu$-calculus in

which *fst*-reduction will be terminating. This alternative calculus will allow us to use termination arguments to conclude the confluence proof of $\boldsymbol{\beta}_{fst}$.

Moreover, the μ-closure condition will be used in an essential way in Lemma 3.7.

3.4 Commutation and Confluence Properties for $\boldsymbol{\beta}_{fst}$-Reduction

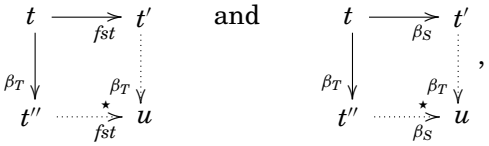
In this section, we prove several commutation and confluence properties related to (subsystems of) $\boldsymbol{\beta}fst$-reduction. The following subsection will provide sufficient arguments to ensure confluence of $\boldsymbol{\beta}fst$-reduction in Proposition 3.22.

LEMMA 3.6.

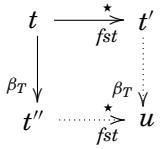
(1) *β_T and $\boldsymbol{\beta}fst$ commute.*
(2) *fst and $\beta_T fst$ commute.*

PROOF.

(1) β_T commutes with β_T, *fst* and β_S; Indeed, one has:
—β_T/β_T : since β_T is confluent in $\Lambda\mu$ (identical to λ-calculus case).
—β_T/fst and β_T/β_S : there is no critical pair between β_T and *fst* (respectively β_T and β_S), only redex *fst* (respectively, β_S) may be duplicated (or erased) by β_T and then:



which ensures commutation.
Lemma 3.3 allows to conclude that β_T and $\boldsymbol{\beta}fst$ commute.
(2) The following holds:
—*fst*/*fst*: *fst* is strongly confluent;
—fst/β_T: the following commutation diagram holds:



Lemma 3.3 allows to conclude *fst* and $\beta_T fst$ commute. □

Next, we prove a weak commutation result relating *fst* and β_S. This will be needed to prove commutation of *fst* and $\boldsymbol{\beta}fst$ in Lemma 3.9. There is no commutation property involving *fst* and β_S since there is a critical pair that

needs extra-rules to converge:

$$\begin{array}{ccc} \mathcal{C}[(\mu\alpha.t)\beta] & \xrightarrow[fst]{} & \mathcal{C}[(\lambda x.\mu\gamma.t\{(u)x\gamma/(u)\alpha\})\beta] \\ \downarrow{\scriptstyle \beta_S} & & \\ \mathcal{C}[t\{\beta/\alpha\}] & & \end{array}$$

This critical pair is convergent because the term on which the reduction begins has no free stream variable (it is μ-closed). As a conclusion, there is a μ-abstraction that binds variable β and $\mathcal{C}$ may be decomposed as: $\mathcal{C}[] = \mathcal{C}_1[\mu\beta.\mathcal{C}_2[]]$.

We apply reduction *fst* to this abstraction in $\mathcal{C}_1[\mu\beta.\mathcal{C}_2[t\{\beta/\alpha\}]]$ that results in $\mathcal{C}_1[\lambda z.\mu\delta.(\mathcal{C}_2[t\{\beta/\alpha\}]\{(u)z\delta/(u)\beta\})] = \mathcal{C}_1[\lambda z.\mu\delta.(\mathcal{C}_2[t\{(u)z\delta/(u)\alpha\}]\{(u)z\delta/(u)\beta\})]$. One then applies the sequence $fst; \beta_T; \beta_S$ to $\mathcal{C}_1[\mu\beta.\mathcal{C}_2[(\lambda x.\mu\gamma.t\{(u)x\gamma/(u)\alpha\})\beta]]$ in order to obtain the following diagram.

$$\begin{array}{ccc} \mathcal{C}[(\mu\alpha.t)\beta] & \xrightarrow[fst]{} & \mathcal{C}_1[\mu\beta.\mathcal{C}_2[(\lambda x.\mu\gamma.t\{(u)x\gamma/(u)\alpha\})\beta]] \\ \downarrow{\scriptstyle \beta_S} & & \downarrow{\scriptstyle fst} \\ & & \mathcal{C}_1[\lambda z.\mu\delta.(\mathcal{C}_2[(\lambda x.\mu\gamma.t\{(u)x\gamma/(u)\alpha\})z\delta]\{(u)z\delta/(u)\beta\})] \\ & & \downarrow{\scriptstyle \beta_T} \\ & & \mathcal{C}_1[\lambda z.\mu\delta.(\mathcal{C}_2[(\mu\gamma.t\{(u)z\gamma/(u)\alpha\})\delta]\{(u)z\delta/(u)\beta\})] \\ & & \downarrow{\scriptstyle \beta_S} \\ \mathcal{C}_1[\mu\beta.\mathcal{C}_2[t\{\beta/\alpha\}]] & \xrightarrow[fst]{} & \mathcal{C}_1[\lambda z.\mu\delta.(\mathcal{C}_2[t\{(u)z\delta/(u)\alpha\}]\{(u)z\delta/(u)\beta\})] \end{array}$$

One notices that the instances of rule β_T used in this reduction are particularly simple: they substitute only variables (this is β_T^{var}). Such instances of η_T cannot duplicate a redex. This last point shall later be crucial for the proof.

LEMMA 3.7.

$$\begin{array}{ccc} t & \xrightarrow[fst]{\star} & t' \\ \downarrow{\scriptstyle \beta_S}\star & & \downarrow{\scriptstyle \beta^{var} fst}\star \\ t'' & \xrightarrow[fst]{\star} & u \end{array}$$

PROOF. When $\begin{array}{ccc} t & \xrightarrow[fst]{} & t' \\ \downarrow{\scriptstyle \beta_S} & & \\ t'' & & \end{array}$, there are three possibilities.

(1) In the case that has been discussed prior to the lemma (i.e., t contains a μ-abstraction $\mu\alpha$ on which there is both a *fst*-reduction to t' and a β_S-reduction to t''), the diagram converges by using respectively a *fst*-reduction from t'' and a sequence $fst; \beta_T^{var}; \beta_S$ from t'.

(2) In the dual case (variable β that undergoes the *fst*-reduction to t' is the argument of a β_S-redex leading to t''), one has:

$$\begin{array}{ccc} \mathcal{C}_1[\mu\beta.\mathcal{C}_2[(\mu\alpha.t)\beta]] & \xrightarrow[fst]{} & \mathcal{C}_1[\lambda x.\mu\gamma.(\mathcal{C}_2[(\mu\alpha.t)x\gamma]\{(u)x\gamma/(u)\beta\})] \\ \downarrow{\scriptstyle \beta_S} & & \\ \mathcal{C}_1[\mu\beta.\mathcal{C}_2[t\{\beta/\alpha\}]] & & \end{array}$$

the diagram converges in a similar way to the previous case:
—one applies a *fst*-reduction at $\mu\beta$ in t'' and
—a sequence $fst; \beta_T^{var}; \beta_S$ in t':

(a) the *fst*-redex is reduced to $\mu\alpha$, creating a β_T^{var}-redex in the term

$$\mathcal{C}_1[\lambda x.\mu\gamma.(\mathcal{C}_2[(\lambda z.\mu\delta.t\{(u)z\delta/(u)\alpha\})x\gamma]\{(u)x\gamma/(u)\beta\})];$$

(b) this β_T^{var}-redex is then reduced, this creates a β_S-redex and finally
(c) one finishes the reduction sequence by a β_S-reduction producing a term $\mathcal{C}_1[\lambda x.\mu\gamma.(\mathcal{C}_2[t\{(u)x\gamma/(u)\alpha\}]\{(u)x\gamma/(u)\beta\})]$

which is precisely the term obtained by the *fst*-reduction from t''.

(3) Last, there is the case when the stream variable undergoing reduction *fst* is not in an argument position in the β_S-redex and when the two redexes do not share the same μ-abstraction, which allows to have a strong commutation diagram.

$$\begin{array}{ccc} t & \xrightarrow[fst]{} & t' \\ \downarrow{\scriptstyle \beta_S} & & \downarrow{\scriptstyle \beta_S} \\ t'' & \xrightarrow[fst]{} & u \end{array}$$

Thus, one has the following diagram:

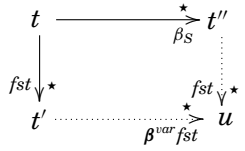
$$\begin{array}{ccccc} t & & \xrightarrow[\beta_S]{} & & t'' \\ \downarrow{\scriptstyle fst} & & & & \downarrow{\scriptstyle fst} \\ t' & \xrightarrow[fst]{=} & \xrightarrow[\beta_T^{var}]{=} & \xrightarrow[\beta_S]{} & u \end{array}$$

Strong confluence of *fst* and strong commutation[4] of *fst* and β_T^{var} ensure that one may have commutation of an arbitrarily long sequence of *fst*-reduction with a β_S-reduction.

$$\begin{array}{ccccc} t & & \xrightarrow[\beta_S]{} & & t'' \\ \downarrow{\scriptstyle fst}\star & & & & \downarrow{\scriptstyle fst}\star \\ t' & & \xrightarrow[fst\beta_T^{var}]{\star} & & u \end{array}$$

[4]Commutation of *fst* and β_T (see Lemma 3.6) becomes strong commutation when considering β_T^{var} since the *fst*-redex cannot be duplicated nor erased.

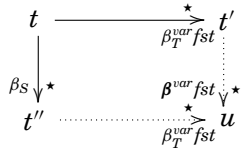
Finally, confluence of the following diagram is obtained by a trivial induction on the length of the β_S-reduction from t to t'':



from which we conclude. □

The following lemma is a simple consequence of $\beta_T^{var}/\boldsymbol{\beta}^{var}{}_{fst}$ commutation and of the previous lemma.

LEMMA 3.8.



LEMMA 3.9.

(1) *fst and* $\boldsymbol{\beta}$*fst commute;*
(2) β_T*fst and* $\boldsymbol{\beta}$*fst commute;*
(3) β_T*fst and* $\boldsymbol{\beta}^{var}$*fst commute.*

PROOF.

(1) It is an immediate consequence of Lemmas 3.6 and 3.7 by applying Lemma 3.4 (variant of the commutation of the union of reductions) in order to obtain the desired commutation.
(2) Once more, it is enough to apply Lemma 3.3: since β_T and *fst* commute with $\boldsymbol{\beta}$*fst*, their union β_T*fst* commutes with $\boldsymbol{\beta}$*fst*.
(3) The last part of the lemma is obtained thanks to Lemma 3.3 as a corollary of the commutation of β_T (respectively *fst*) with $\boldsymbol{\beta}^{var}$*fst* which is itself obtained as for Lemma 3.6 (respectively as for the first point of the present lemma). □

3.5 Indexed $\Lambda\mu$-Terms

In the previous section, we established several commutation and confluence properties of subsystems of $\boldsymbol{\beta}$*fst* but confluence of $\boldsymbol{\beta}$*fst* is not yet achieved. Indeed, as we saw at Lemma 3.7, commutation of *fst* with β_S is slightly problematic since one gets commutation only through a larger reduction using *fst* are β_T^{var} (for terms t, t', t'' such that $t \to_{fst} t'$ and $t \to_{\beta_S} t''$, there exists u such that $t' \to_{(fst;\beta_T^{var})^=;\beta_S} u$ and $t'' \to_{fst} u$).

As a consequence, the proof of the fact that $\boldsymbol{\beta}^{var}{}_{fst}$ commutes with $\boldsymbol{\beta}_{fst}$ (i.e., Lemma 3.21) is delicate. In order to achieve this commutation, one will use a variant of *fst* which is terminating. As a consequence, one will be considering

three terminating reductions (β_S, β_T^{var}, and the variant of *fst*, called fst_ι), which will result in great simplification of the commutation results.

Obtaining a terminating *fst*-reduction is achieved by annotating $\Lambda\mu$-terms with a *fst-expansion potential*. One considers the following variants of $\Lambda\mu$-calculus:

Definition 3.10. Let i have values among the (positive or negative) integers. One considers terms built on the following syntax:

$$t, u ::= x \mid \lambda x.t \mid \mu^i\alpha.t \mid (t)\alpha \mid (t)u$$

Only the *fst*-reduction is impacted by this change: *fst* is turned to $_{fst_\iota}$ which can only be applied if the index on μ, also referred to as its *potential*, is strictly positive and which decreases this index by one:

$$\mu^{i+1}\alpha.t \longrightarrow_{fst_\iota} \lambda x.\mu^i\beta.t\{(u)x\beta/(u)\alpha\} \qquad \text{with } i \geq 0.$$

The other rules are unchanged: for instance, reduction β_S becomes

$$(\mu^i\alpha.t)\beta \longrightarrow_{\beta_S} t\{\beta/\alpha\}.$$

The indexes, if badly chosen, could cause failure of confluence of $\boldsymbol{\beta}^{var}{}_{fst_\iota}$ due to problems with β_S-reduction. Indeed if one considers term $t = \mu^1\alpha.(\mu^0\beta.x)\alpha$, then one has non-confluence since t can reduce to $\lambda y.\mu^0\alpha.(\mu^0\beta.x)y\alpha$ using a $_{fst_\iota}$-reduction and to $\lambda y.\mu^0\alpha.x$ using a β_S-reduction followed by a $_{fst_\iota}$-reduction and both terms are $\boldsymbol{\beta}^{var}{}_{fst_\iota}$-normal forms. The problem comes from the fact that a β_S-reduction may cause communication of stream variables which have not the same potential.

We shall thus consider a subset of the indexed calculus for which the μ-indexes are coherent with respect to the $\boldsymbol{\beta}^{var}{}_{fst}$-reduction (but not with full β_T-reduction which is of no interest at that point):

Definition 3.11 (*Indexing a $\Lambda\mu$-Term*). Let t be a $\Lambda\mu$-term with free stream variables among V. One first annotates every sub-term of t at an argument position by using lists of integers l' (of the form: $(u)v^{l'}$), obtaining t'. Then, one considers a partial function ρ from stream variables to integers, which is defined for all elements in V, and a list of integers l.

The ***indexed term*** for t, $[t']^l_\rho$, is obtained from t' as follows:

$$\begin{array}{lll}
[x]^l_\rho = x; & & [(t)\alpha]^l_\rho = ([t]^{\rho(\alpha)::l}_\rho)\alpha; \\
[\lambda x.t]^{n::l}_\rho = \lambda x.[t]^{(n-1)::l}_\rho; & & [\lambda x.t]^{\emptyset}_\rho = \lambda x.t; \\
[(t)u^{l'}]^{n::l}_\rho = ([t]^{(n+1)::l}_\rho)[u]^{l'}_\rho; & & [(t)u^{l'}]^{\emptyset}_\rho = (t)u; \\
[\mu\alpha.t]^{n::l}_\rho = \mu^n\alpha.[t]^l_{\rho\cup\{\alpha\leftarrow n\}}; & & [\mu\alpha.t]^{\emptyset}_\rho = \mu\alpha.t.
\end{array}$$

Definition 3.12 (*Indexed $\Lambda\mu$-Term, $\Lambda\mu^\iota$-Term*). An *indexed* $\Lambda\mu$-*term* t is a term of the indexed syntax which comes from the translation of a closed $\Lambda\mu$-term ($t = [u']^l_\rho$ for some u', ρ, l, u' being obtained by annotating the subterms of a $\Lambda\mu$-term u which are in argument positions by lists of integers).

One shall also speak of $\Lambda\mu^\iota$-*terms* to refer to the indexed $\Lambda\mu$-terms.

PROPOSITION 3.13 (POSITIVE INDEXING). *To any $\Lambda\mu$-term t, one can associate an indexing ι such that the resulting t^ι is a $\Lambda\mu^\iota$-term and all μ have positive indexes.*

PROOF. It is sufficient to choose long enough lists of large enough integers. □

PROPOSITION 3.14 (INDEXING PRESERVATION). *If t is a μ-closed $\Lambda\mu^\iota$-term and if $t \longrightarrow_{\boldsymbol{\beta}^{var} fst_\iota} u$, then u is also a $\Lambda\mu^\iota$-term.*

PROOF. One simply checks that the three rules β_T, β_S and fst_ι preserve the property of be the result of an indexing of a $\Lambda\mu$-term.

—*Case β_S.* Let t^ι be a $\Lambda\mu^\iota$-term obtained by indexing a $\Lambda\mu$-term t and let us suppose that $t^\iota \longrightarrow_{\beta_S} v$. There exist $\mathcal{C}$ and t' such that $t^\iota = \mathcal{C}^\iota[(\mu^i\alpha.t')\beta]$ and $v = \mathcal{C}^\iota[t'\{\beta/\alpha\}]$. By Definition 3.11, in t^ι, the binder for β has the same index i as $\mu^i\alpha.t$ and, v is obtained by indexing a $\Lambda\mu$-term u;

—*Case β_T^{var}.* Let t^ι be a $\Lambda\mu^\iota$-term obtained by indexing a $\Lambda\mu$-term t and let us suppose that $t^\iota \longrightarrow_{\beta_T^{var}} v$. There exist $\mathcal{C}$ and t' such that $t^\iota = \mathcal{C}^\iota[(\lambda x.t')y]$ and $v = \mathcal{C}^\iota[t'\{y/x\}]$. Let us consider the definition of the indexing of a $\Lambda\mu$-term, one notices that: $[(\lambda x.t')y^{l'}]_\rho^{n::l} = ([\lambda x.t']_\rho^{(n+1)::l})y = (\lambda x.[t']_\rho^{n::l})y$ et $[t'\{y/x\}]_\rho^{n::l} = [t']_\rho^{n::l}\{y/x\}$.

—*Case fst_ι.* Let t^ι be a $\Lambda\mu^\iota$-term obtained by indexing a $\Lambda\mu$-term t and let us suppose that $t^\iota \longrightarrow_{fst_\iota} v$. Suppose that t^ι is of the form $\mathcal{C}^\iota[\mu^{i+1}\alpha.t']$, one has $v = \mathcal{C}^\iota[\lambda x.\mu^i\alpha.t'\{(u)x\alpha/(u)\alpha\}]$. One easily checks that compatibility is maintained by adding variable x as an argument of all subterms u to which was applied a stream variable α. □

PROPOSITION 3.15 ($\boldsymbol{\beta}^{var} fst_\iota$-TERMINATION). *$\boldsymbol{\beta}^{var} fst_\iota$-reduction terminates for indexed $\Lambda\mu$-terms.*

PROOF. This is immediate since $\boldsymbol{\beta}^{var}$ terminates and each occurrence of fst_ι lower the index of one of the μ-abstractions by 1. □

PROPOSITION 3.16 (COMPATIBLE INDEXINGS). *If $t, t_1, \ldots, t_n$ are $\Lambda\mu$-terms such that $t \longrightarrow^\star_{\boldsymbol{\beta}^{var} fst} t_j$ for any $1 \leq j \leq n$, there exist indexings $\iota, \iota_1, \ldots \iota_n$ for $t, t_1 \ldots t_n$ such that all μ-abstractions in the t_j have positive indexes and that the reductions $t \longrightarrow^\star_{\boldsymbol{\beta}^{var} fst} t_j$ can be realized within $\boldsymbol{\beta}^{var} fst_\iota$.*

PROOF. One proves the result by induction on the number n of terms t_i. The base case is immediate by providing an indexing to t_1 as a $\Lambda\mu^\iota$-term with all positive indexes and by following back the reduction sequence $t \longrightarrow^\star_{\boldsymbol{\beta}^{var} fst} t_1$ in order to obtain a indexing for t which is compatible with the considered reduction sequence. The inductive step is obtained by noticing that if a $\boldsymbol{\beta}^{var} fst_\iota$-reduction r is compatible with an indexing of t, then all indexings providing larger indexes or using longer lists are compatible with reduction r. □

Moreover, the coherence constraint on indexes of $\Lambda\mu^\iota$-terms ensures that all commutation or confluence diagrams that one can close in $\boldsymbol{\beta}^{var} fst$ can also be closed in $\boldsymbol{\beta}^{var} fst_\iota$ (for instance β_S and $\boldsymbol{\beta}^{var}$ are strongly confluent). More

generally, one shall refer to the adaptation of a lemma on $\Lambda\mu$-terms to $\Lambda\mu^\iota$-terms by adding the letter ι to its label. For instance:

LEMMA 3.17 (3.7.(ι)). *If t, t', t'' are $\Lambda\mu^\iota$-terms such that $t \longrightarrow^\star_{fst_\iota} t'$ and $t \longrightarrow^\star_{\beta_S} t''$, there exists u such that $t' \longrightarrow^\star_{\boldsymbol{\beta}^{var} fst_\iota} u$ and $t'' \longrightarrow^\star_{fst_\iota} u$.*

and

LEMMA 3.18 (3.8.(ι)). *If t, t', t'' are $\Lambda\mu^\iota$-terms such that $t \longrightarrow^\star_{\beta_T^{var} fst_\iota} t'$ and $t \longrightarrow^\star_{\beta_S} t''$, there exists u such that $t' \longrightarrow^\star_{\boldsymbol{\beta}^{var} fst_\iota} u$ and $t'' \longrightarrow^\star_{\beta_T^{var} fst_\iota} u$.*

PROOF. Proofs are very much similar to those of Lemmas 3.7 and 3.8. □

In the following lemma, one shall write $\longrightarrow_1$ for $\longrightarrow_{\beta_S}$ and $\longrightarrow_2$ for $\longrightarrow_{\beta_T^{var} fst_\iota}$.

LEMMA 3.19 (3.20.(ι)). *If t is a $\Lambda\mu^\iota$-term and t', t'' are such that $t \longrightarrow^\star_{1,2} t'$ and $t \longrightarrow^\star_{1,2} t''$, then there exists a $\Lambda\mu^\iota$-term u such that $t' \longrightarrow^\star_{1,2} u$ and $t'' \longrightarrow^\star_{1,2} u$.*

PROOF. One shall first notice that t' and t'' are $\Lambda\mu^\iota$-terms, as well as all terms occurring on the reduction sequences $t \longrightarrow^\star_{1,2} t'$ and $t \longrightarrow^\star_{1,2} t''$. The result is then shown by induction on the length of a longest derivation of $\boldsymbol{\beta}^{var}{}_{fst_\iota}$ from t, $lg_\iota(t)$ (it exists thanks to termination of $\boldsymbol{\beta}^{var}{}_{fst_\iota}$).

(1) If $lg_\iota(t) = 0$, the result is immediate: there is no reduction and $t = t' = t'' = u$.
(2) Let us suppose the result true for terms t such that $lg_\iota(t) \leq n$ for a certain integer n. One shows that if t is such that $lg_\iota(t) = n + 1$, then the result is also true for t. Indeed, let t' and t'' be such that $t \longrightarrow^\star_{1,2} t'$ and $t \longrightarrow^\star_{1,2} t''$. If $t = t'$ or $t = t''$, the result is trivial. Otherwise, it is the case that $t \longrightarrow_{1,2} t'_1 \longrightarrow^\star_{1,2} t'$ and $t \longrightarrow_{1,2} t''_2 \longrightarrow^\star_{1,2} t''$. Let us reason by case on $t \longrightarrow_{1,2} t'_1$ and $t \longrightarrow_{1,2} t''_1$ and let us prove that in any case, there exists u_0 such that $t'_1 \longrightarrow^\star_{1,2} u_0$ and $t''_1 \longrightarrow^\star_{1,2} u_0$. The only case that deserves to be detailed is when there is a critical pair between β_S and fst_ι for which existence of u_0 is obtained by Lemma 3.7.(ι). Since $lg_\iota(t'_1), lg_\iota(t''_1) \leq n$ one may apply the induction hypothesis to t''_1 so that one concludes that there exists u_1 such that $t'' \longrightarrow^\star_{1,2} u_1$, $u_0 \longrightarrow^\star_{1,2} u_1$. One has $t'_1 \longrightarrow^\star_{1,2} u_0 \longrightarrow^\star_{1,2} u_1$, which allows us to apply the induction hypothesis to t'_1 and to conclude that there exists a term u such that $u_1 \longrightarrow^\star_{1,2} u$ and $t' \longrightarrow^\star_{1,2} u$.
    Finally, one has $t' \longrightarrow^\star_{1,2} u$ and $t'' \longrightarrow^\star_{1,2} u_1 \longrightarrow^\star_{1,2} u$ and the situation is summarized in Figure 3.

As a conclusion, for any $\Lambda\mu^\iota$-terms, t, t', t'' such that $t \longrightarrow^\star_{1,2} t'$ and $t \longrightarrow^\star_{1,2} t''$, there exists a $\Lambda\mu^\iota$-term u such that $t' \longrightarrow^\star_{1,2} u$ and $t'' \longrightarrow^\star_{1,2} u$. □

Lemma 3.19 allows to conclude the confluence of $\boldsymbol{\beta}^{var}{}_{fst}$.

LEMMA 3.20 ($\boldsymbol{\beta}^{var}{}_{fst}$-REDUCTION IS CONFLUENT). *Let $t, t', t'' \in \Sigma^c_{\Lambda\mu}$.*
If $t \longrightarrow^\star_{\boldsymbol{\beta}^{var} fst} t', t''$, then there exists u such that $t', t'' \longrightarrow^\star_{\boldsymbol{\beta}^{var} fst} u$.

PROOF. Let $(-)^\iota$ an indexing for term t that is compatible with t' and t'': $(t)^\iota \longrightarrow^\star_{\boldsymbol{\beta}^{var} fst_\iota} (t')^\iota, (t'')^\iota$. From Lemma 3.20($\iota$), there exists an indexed $\Lambda\mu$-term

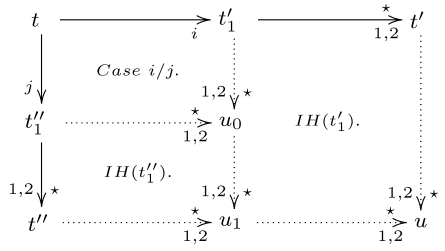


Fig. 3. Confluence diagram for Lemma 3.20.ι.

v such that $(t')^\iota, (t'')^\iota \longrightarrow^\star_{\beta^{var}fst_\iota} v$. Let u be the $\Lambda\mu$-term obtained by erasing the indexes in v, then $t', t'' \longrightarrow^\star_{\beta^{var}fst} u$. □

Commutation of $\beta^{var}fst$ and βfst and confluence of βfst follow.

LEMMA 3.21. *$\beta^{var}fst$ and βfst commute.*

PROOF. $\beta^{var}fst$ commutes with $\beta_T fst$ commute (Lemma 3.9) and with $\beta^{var}fst$ (Lemma 3.20). By Lemma 3.3, $\beta^{var}fst$ commutes with βfst. □

PROPOSITION 3.22. *Reduction βfst is confluent.*

PROOF. The union of reductions $\beta^{var}_T \beta_S fst$ et $\beta_T fst$ is actually reduction βfst which results in commutation of βfst and βfst that is confluence of βfst. □

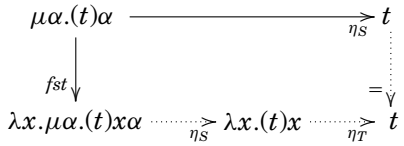
3.6 Commutation of η and βfst.

PROPOSITION 3.23. *η is confluent.*

PROOF. η is actually strongly confluent. □

PROPOSITION 3.24. *η commutes with βfst.*

PROOF. This is proved thanks to the commutation of η with, respectively, β_S, β_T and fst which are all three elementary. For instance:



Finally, one gets commutation of η with βfst by Lemma 3.3. □

The proof of confluence theorem for $\Lambda\mu$-calculus can thus be concluded as follows:

PROOF OF CONFLUENCE THEOREM. Thanks to confluence of βfst (Proposition 3.22), confluence of η (Proposition 3.23) and commutation of the two previous systems (Proposition 3.24), we can apply Hindley-Rosen lemma to obtain confluence of $\Lambda\mu$-calculus on μ-closed $\Lambda\mu$-terms. □

3.7 Confluence of $\lambda\mu\eta$-Calculus

$\lambda\mu\eta$-calculus is stable by $\Lambda\mu$-reductions.

LEMMA 3.25. *If $t \in \Sigma_{\lambda\mu}$ and $t \to^{\star}_{\Lambda\mu} u$, then $u \in \Sigma_{\lambda\mu}$ and $t \to^{\star}_{\lambda\mu\eta} u$.*

Confluence of $\lambda\mu\eta$-calculus is thus easily deduced from confluence of $\Lambda\mu$-calculus:

COROLLARY 3.26. *$\lambda\mu\eta$-calculus is confluent on μ-closed terms.*

Remark 3.27. A proof of confluence for $\lambda\mu\eta$-calculus can be found in Py's thesis [Py 1998] and is outlined in David and Py [2001]. Our proof of confluence for $\Lambda\mu$-calculus is simpler, in particular we can avoid a lengthy development where Py uses annotations on terms. The simplification lies in particular in the use of Lemma 3.7, the indexing and the corresponding termination arguments and in the fact that we avoid redundancy among $\Lambda\mu$-reductions by not considering μ as part of the reductions of $\Lambda\mu$-calculus.

3.8 $\Lambda\mu$-Calculus is Church-Rosser

Usually, Church-Rosser property (if two terms are equivalent, then they can both be reduced to a common term) is an immediate consequence of confluence. However, it is not the case here since even though t, u are μ-closed, the sequence of terms justifying $t =_{\Lambda\mu} u$ may involve nonclosed terms for which confluence does not hold in general. As a consequence, the following proposition is not a trivial consequence of confluence theorem for $\Lambda\mu$:

PROPOSITION 3.28. *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 $t, u \longrightarrow^{\star}_{\Lambda\mu} v$.*

PROOF. Let t and u be two closed $\Lambda\mu$-terms such that $t =_{\Lambda\mu} u$. Then, there exist $(t_i)_{0\leq i\leq n}$, $(u_i)_{0\leq i\leq n}$ such that $t = t_0$, $u = u_n$ and for $0 \leq i \leq n$, $t_i \longrightarrow^{\star}_{\Lambda\mu} u_i$, for $1 \leq i \leq n$, $t_{i+1} \longrightarrow^{\star}_{\Lambda\mu} u_i$.

We reason as follows: let $\alpha_1, \cdots, \alpha_n$ be the stream variables occurring free in terms $(t_i)_{0\leq i\leq n}, (u_i)_{0\leq i\leq n}$ and let $\mathcal{C}[] = \mu\alpha_1 \ldots \alpha_n.[]$. Consider $t' = \mathcal{C}[t]$ and $u' = \mathcal{C}[u]$, as well as $t'_i = \mathcal{C}[t_i]$ and $u'_i = \mathcal{C}[u_i]$ for $0 \leq i \leq n$.

It is clear that $t' =_{\Lambda\mu} u'$ and all t'_is and u'_is are μ-closed terms, which allows us to apply Theorem 3.1.

By an easy induction, we obtain that for all $0 \leq i \leq n$, there exists v'_i such that $u'_0 \longrightarrow^{\star}_{\Lambda\mu} v'_i \longleftarrow^{\star}_{\Lambda\mu} u'_i$. One then deduces the existence of v' such that $t' \longrightarrow^{\star}_{\Lambda\mu} v' \longleftarrow^{\star}_{\Lambda\mu} u'$ (it is actually term v'_n), that is such that:

$$\mathcal{C}[t] \longrightarrow^{\star}_{\Lambda\mu} v' \longleftarrow^{\star}_{\Lambda\mu} \mathcal{C}[u]$$

Let ρ_1 and ρ_2 be the reduction sequences from t' to v' and from u' to v' respectively. Since t and u have no free stream variables, variables $\alpha_1, \ldots, \alpha_n$ which are abstracted in the context $\mathcal{C}[]$ have no occurrence neither in t nor in u. One finally obtains that every pair of terms v_1, v_2 occurring respectively in ρ_1, ρ_2 are of the form $\mathcal{C}_1[t_1]$ and $\mathcal{C}_2[u_2]$ with $t \longrightarrow^{\star}_{\Lambda\mu} t_1$ and $u \longrightarrow^{\star}_{\Lambda\mu} u_2$, and the contexts $\mathcal{C}_1[]$ and $\mathcal{C}_2[]$ are obtained by adding some λ-abstractions before some of the μ-abstractions, resulting from the *fst*-reductions which are applied in ρ_1 and ρ_2.

One thus has $\mathcal{C}[t] \longrightarrow^{\star}_{\Lambda\mu} \mathcal{C}_t[t''] = v' = \mathcal{C}_u[u''] \longleftarrow^{\star}_{\Lambda\mu} \mathcal{C}[u]$, with $t \longrightarrow^{\star}_{\Lambda\mu} t''$ and $u \longrightarrow^{\star}_{\Lambda\mu} u''$.

To conclude, one notices that $\mathcal{C}_t = \mathcal{C}_u$: indeed, the number of μ-abstractions in the $\mathcal{C}[]$ derived prefixes never changes and terms $\mathcal{C}_t[t'']$ and $\mathcal{C}_u[u'']$ being equal, they have the same prefixes: $\mathcal{C}_t = \mathcal{C}_u$ which implies: $t'' = v = u''$. We can finish the proof of the proposition by concluding that $t \longrightarrow^{\star}_{\Lambda\mu} v \longleftarrow^{\star}_{\Lambda\mu} u$. □

3.9 Characterizing Canonical Normal Forms in $\Lambda\mu$

The following proposition is evidently equivalent to the definition of canonical normal forms in the μ-closed case.

PROPOSITION 3.29. *μ-closed canonical normal forms are exactly the μ-closed $\Lambda\mu$-terms in $\beta\eta fst^{-}$-normal form.*

The following proposition corresponds to the property of uniqueness of the $\beta\eta$-normal form in λ-calculus:

PROPOSITION 3.30. *μ-closed canonical normal forms are $\Lambda\mu$-equivalent if, and only if, they are fst-equivalent:*

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

PROOF. Let t and u be two μ-closed $\Lambda\mu$-terms in canonical normal form such that $t =_{\Lambda\mu} u$. Thanks to Proposition 3.28, we know that there exists v such that $t, u \longrightarrow^{\star}_{\Lambda\mu} v$. By hypothesis, the only $\Lambda\mu$-redexes that can occur in t and u are *fst*-redexes and reducing them only creates new *fst*-redexes.

As a conclusion we have $t, u \longrightarrow^{\star}_{fst} v$ and finally $t =_{fst} u$. □

The previous result characterizes more precisely the canonical normal forms in $\Lambda\mu$-calculus. As a conclusion, one knows that separation in $\Lambda\mu$-calculus is up to *fst*-equivalence. We can actually be even more precise. Indeed, given a canonical normal form t and its *fst*-equivalence class $\underline{t}$ (of canonical normal forms), there is a minimal element which can be taken as a representative of the equivalence class:

PROPOSITION 3.31. *Let $t \in \Sigma^{c}_{\Lambda\mu}$ be a in canonical normal form and $\underline{t}$ be its fst-equivalence class. Then, there exists a term $\underline{t}_{min}$ which is minimal in $\underline{t}$ with respect to fst:*

$$\forall u \in \underline{t}\ ,\ \ \underline{t}_{min} \longrightarrow^{\star}_{fst} u.$$

PROOF. Consider reduction fst^{-1} defined as $t \longrightarrow_{fst^{-1}} u$ is $u \longrightarrow_{fst} t$. fst^{-1} is confluent (it has no critical pair) and normalizing (since the size of terms strictly decreases on a fst^{-1} reduction sequence). $\underline{t}_{min}$ is thus the unique normal form for fst^{-1} in $\underline{t}$. □

Definition 3.32 (*Minimal canonical terms*). Let $t \in \Sigma^{c}_{\Lambda\mu}$. t is *minimal canonical* if t is in canonical normal form and $t = \underline{t}_{min}$.

As a consequence, Theorem 2.10 actually can be enforced to state that every pair of syntactically distinct minimal canonical terms can be separated.

	Confluence	Separation	Type System	Subject Red.	Strong Norm.
$\lambda\mu$	yes	no	CND	yes	yes
$\lambda\mu\eta$	yes (μ-closed)	no	CND + $\bot$	yes	yes
$\lambda\mu\epsilon$	yes	?	CND + $\bot$	yes	yes
$\Lambda\mu$	yes (μ-closed)	yes	CND + $\bot$ and $\Lambda_{\mathcal{S}}$	yes	yes

Fig. 4. Properties of the four CBN $\lambda\mu$-calculi.

THEOREM 3.33. *Let $t, u \in \Sigma^c_{\Lambda\mu}$. If t, u are minimal canonical and are syntactically different, then there exists a stream applicative context $\mathcal{C}$ such that*

$$\mathcal{C}[t] \longrightarrow^{\star}_{\Lambda\mu} 1 \quad \textit{and} \quad \mathcal{C}[u] \longrightarrow^{\star}_{\Lambda\mu} 0.$$

Remark 3.34. It is immediate, after Definition 2.12, that $=_{\Lambda\mu}==_{\Lambda\mu^-}$. However we do not consider the reduction system $\Lambda\mu^-$ for $\Lambda\mu$-calculus since it is not confluent. Indeed, there are different canonical normal forms that are $=_{\Lambda\mu}$-equivalent (and thus $=_{\Lambda\mu^-}$-equivalent) but that are normal forms for $\longrightarrow_{\Lambda\mu^-}$, which contradicts confluence of $\Lambda\mu^-$.

3.10 Comparing $\lambda\mu$-Calculi

In this section, we review some of the known properties of the calculi and we investigate their relationships.

The four calculi introduced in Section 2 share some properties but differ with respect to other properties. We summarize in Figure 4 the properties of the four $\lambda\mu$-calculi considered in the first section of the article. The four calculi satisfy confluence [Parigot 1992; Py 1998]. They all have a type system that is in correspondence with classical natural deduction proofs, with or without explicit $\bot$. In the next section of the article we shall introduce and study $\Lambda_{\mathcal{S}}$, a new type system for $\Lambda\mu$-calculus. Strong normalization is known in the simply typed case for $\lambda\mu\eta$, $\lambda\mu\epsilon$ and $\Lambda\mu$ while Parigot [1997] provided a proof of strong normalization for second-order $\lambda\mu$.

3.10.1 *Conservative Extensions.* Thanks to Lemma 3.25 and Theorem 3.1, one easily obtains that $\Lambda\mu$ is a conservative extension of $\lambda\mu\eta$.

PROPOSITION 3.35 ($\Lambda\mu$ IS A CONSERVATIVE EXTENSION OF $\lambda\mu\eta$). *If $t, u \in \Sigma^c_{\lambda\mu}$, then $t =_{\Lambda\mu} u \Leftrightarrow t =_{\lambda\mu\eta} u$.*

PROOF. It is immediate that if $t, u \in \Sigma^c_{\lambda\mu}$ then $t =_{\lambda\mu\eta} u$ implies $t =_{\Lambda\mu} u$. The converse property requires confluence and Proposition 3.28: by confluence of $\Lambda\mu$ on $\Sigma^c_{\Lambda\mu}$, if $t =_{\Lambda\mu} u$ there exists $v \in \Sigma^c_{\Lambda\mu}$ such that $t \to^{\star}_{\Lambda\mu} v \leftarrow^{\star}_{\Lambda\mu} u$. By Lemma 3.25, if moreover $t, u \in \Sigma_{\lambda\mu}$ then there exists $v \in \Sigma_{\lambda\mu}$ such that $t \to^{\star}_{\lambda\mu\eta} v \leftarrow^{\star}_{\lambda\mu\eta} u$ and finally $t =_{\lambda\mu\eta} u$. □

In the same way as $\lambda\mu\eta$-calculus was stable by $\Lambda\mu$-reduction in Lemma 3.25, $\lambda\mu$-calculus is stable by $\lambda\mu\epsilon$-reductions.

LEMMA 3.36. *If $t \in \Sigma_{\lambda\mu}$ and $t \to^{\star}_{\lambda\mu\epsilon} u$, then $u \in \Sigma_{\lambda\mu}$ and $t \to^{\star}_{\lambda\mu} u$.*

As a consequence, one also gets the following.

PROPOSITION 3.37 ($\lambda\mu\epsilon$ IS A CONSERVATIVE EXTENSION OF $\lambda\mu$). *If* $t, u \in \Sigma^c_{\lambda\mu}$, *then* $t =_{\lambda\mu\epsilon} u \Leftrightarrow t =_{\lambda\mu} u$.

PROOF. Similar to the previous proof. □

On the other hand, equational theories of $\Lambda\mu$ and $\lambda\mu\epsilon$ are incomparable.

PROPOSITION 3.38 ($=_{\Lambda\mu}$ AND $=_{\lambda\mu\epsilon}$ ARE INCOMPARABLE). *There exist* $t, u, v \in \Sigma^c_{\Lambda\mu}$ *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 $t \in \Sigma^c_{\Lambda\mu}$ be a term of the form $\mu\alpha.t'$ in canonical normal form with no two consecutive μ-abstractions. Let $u = \lambda x.\mu\alpha.t'\{(w)x\alpha/(w)\alpha\}$ 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 forms 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 in a term. As a conclusion $t \neq_{fst} v$ and finally $t \neq_{\Lambda\mu} v$. □

3.10.2 *Separability Properties.* We showed in a previous work [Saurin 2005, 2008] that $\Lambda\mu$-calculus satisfies 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 [Py 1998; David and Py 2001].

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 anything 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:

$(\mu\alpha.t)u$	$\longrightarrow_{\mu}$	$\mu\alpha.t$	if $\alpha \notin FV(t)$.
$\mu\gamma.(\mu\alpha.t)\beta$	$\longrightarrow_{\rho}$	$\mu\gamma.t$	which is α-equivalent to $\mu\alpha.t$.
$\mu\beta.\mu\alpha.t$	$\longrightarrow_{\epsilon}$	$\mu\beta.\lvert t\rvert_{\alpha} = \mu\beta.t$	which is α-equivalent to $\mu\alpha.t$.

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

4. SIMPLY-TYPED $\Lambda\mu$-CALCULUS

In this section, we introduce a new type system for $\Lambda\mu$-calculus, $\Lambda_{\mathcal{S}}$, and prove some of its properties, namely that it can type strictly more terms than Parigot's type system, that it has subject reduction and that $\Lambda\mu$-calculus typed thanks to $\Lambda_{\mathcal{S}}$ is strongly normalizing.

4.1 Towards $\Lambda_{\mathcal{S}}$

4.1.1 *Typing $\lambda\mu$-Calculus, Classically.* One may, of course think of typing $\Lambda\mu$ 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. Since we have an expansion rule, we restrict the *fst* rule in order to apply it only on terms of type $A \to B$ so that we achieve subject reduction:

$$\mu\alpha^{A\to B}.t \quad \longrightarrow_{fst} \quad \lambda x^A.\mu\beta^B.t\{(u)x\beta/(u)\alpha\}.$$

Even though this type system has good properties, this approach is not satisfying with respect to our original motivation in introducing $\Lambda\mu$-calculus which was to recover separation. Indeed, the constructions that were crucial in the separation proof [Saurin 2005, 2001] are not typable in the classical type system with $\bot$, for very deep reasons. In particular, no term of the shape $\mu\alpha.\lambda x.t$ can be typed in the considered type system although they are typically the kind of terms we need to achieve a Böhm out: for instance parametric pairs $\langle t, u\rangle_k$ are of this form. Actually the type system introduced in order to enrich the Curry-Howard correspondence to classical logic via free deduction [Parigot 1992], as well as the usual extensions considered in the literature, forbid precisely those terms: $\lambda x.t$ is a λ-abstracted term and shall therefore be of an $\to$-type whereas being under a μ-abstraction with stream variable α, it shall be of type $\bot$.

We are thus interested in studying another type system in which some of the computational behaviors that are specific to untyped $\Lambda\mu$-calculus can be observed in the typed setting. In particular, we would think that a suitable type system for $\Lambda\mu$-calculus should allow some form of typed separation as the one that exists for simply typed λ-calculus [Statman 1983; Joly 2000; Došen and Petrić 2000, 2001]. Though typed separation is clearly beyond the scope of the present article, we regard the type system $\Lambda_{\mathcal{S}}$ developed in this article as a good candidate for achieving typed separation in $\Lambda\mu$.

4.1.2 *Making Streams First-Class Citizens in the Typed Setting.* The stream mechanism that was used in the untyped calculus in order to obtain separation is thus unactivated when classical types are reintroduced: one loses the benefit from the extended syntax of $\Sigma_{\Lambda\mu}$. We shall look for a type system that would reflect the stream construction in the types. 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 $\mathcal{S} \Rightarrow \mathcal{T}$ in the following). We can thus think of the following typing rules for μ-constructions:

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

4.1.3 *A Type Mismatch.* However, *fst* reduction does complicate the definition of a type system for $\Lambda\mu$: 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 arrow type $A \to B$ (more precisely $A \to (\mathcal{S}' \Rightarrow \mathcal{T}')$). Moreover, streams and terms should not only be related by the *fst* rule but also by allowing to apply a stream functional to a term (e.g., $(\mu\alpha.t)u$) and conversely, one might want to apply a λ-abstracted term to a stream variable (for instance $(\lambda x.t)\alpha$).

As a conclusion, $\Rightarrow$-types and $\to$-types should be related in some way. The geometry of *fst*-rule provides the key to this connection.

4.1.4 *A Relation Over Stream Types.* *fst* is the synthesis of an η-expansion and a μ-reduction. In the typed case, the η-expansion can occur only on $\to$-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 $(\mathcal{T} \to \mathcal{S}) \Rightarrow \mathcal{T}'$. After an application of *fst*, we have term $\lambda x.\mu\beta.t\{(u)x\beta/(u)\alpha\}$ that should be of type $\mathcal{T} \to (\mathcal{S} \Rightarrow \mathcal{T}')$.

This suggests a sort of an associativity rule between constructors $\to$ and $\Rightarrow$:

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

4.2 Simply Typed Streams: $\Lambda_{\mathcal{S}}$

4.2.1 *Pretypes and Types.* We now define more formally the type system $\Lambda_{\mathcal{S}}$ for $\Lambda\mu$-calculus.

Definition 4.1 ($\Lambda_{\mathcal{S}}$ *Pretypes*). Pretypes are given by the following grammar.

$$\begin{array}{lll} \text{Term pretypes:} & \mathcal{T}, A, B, \ldots & ::= o_i \mid A \to B \mid \mathcal{S} \Rightarrow \mathcal{T} \\ \text{Stream pretypes:} & \mathcal{S}, P, Q, \ldots & ::= \sigma_i \mid \mathcal{T} \to \mathcal{S} \mid \bot \end{array}$$

o_i and σ_i are respectively term and stream type variables. $A_1 \to A_2 \to \cdots \to A_n \to \sigma$ is the type of a stream containing at least n elements. 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.[5] 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 4.2 ($\equiv_{fst}$). $\equiv_{fst}$ is a congruence relation over pretypes, which is the symmetric, reflexive and transitive closure of relation $\succ_{fst}$ defined by

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

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

Definition 4.3 ($\Lambda_{\mathcal{S}}$ *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 4.4 (*Typed* $\Lambda\mu$*-Calculus*). Typed $\Lambda\mu$-terms are given by the following syntax:

$$\boxed{t ::= x \mid \lambda x^{\mathcal{T}}.t \mid (t)u \mid \mu\alpha^{\mathcal{S}}.t \mid (t)\alpha}$$

[5]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. 5. $\Lambda_{\mathcal{S}}$: a type system for $\Lambda\mu$-calculus.

We show in Figure 5 the type system $\Lambda_{\mathcal{S}}$ for $\Lambda\mu$-calculus. In this type system, we deal with pretypes and an explicit conversion rule between two equivalent pre-types.

4.3 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, to allow an application of the *fst* rule on t, we will require that the term has a type represented by a pretype 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

$$\boxed{t : A \to B \longrightarrow_{\eta^{\to}_{exp}} \lambda x^{A}.(t)x : A \to B}.$$

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

Definition 4.5 (*fst*$^{\to}$ *Reduction*). Reduction *fst*$^{\to}$ is defined as a restriction on *fst*-reduction on typed $\Lambda\mu$-terms as follows:

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

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

PROPOSITION 4.6. *Let t, u be two typed $\Lambda\mu$-terms. The following implications hold:*

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

The converse implications do not hold.

PROOF.

—It is clear that $t \longrightarrow_{fst^{\to}} u \quad \Rightarrow \quad t \longrightarrow_{fst} u$;

—The second implication, $t \longrightarrow_{fst^{-}} u \quad \Rightarrow \quad t \longrightarrow_{fst^{\to}} u$, can be proved by noticing that if $t \longrightarrow_{fst^{-}} u$, then either the μ-abstraction to which is applied a *fst*-reduction was of the form: $(\mu\alpha^{\mathcal{S}}.t)u$, or the stream variable α that was bound by this μ-abstraction had an occurrence in a subterm of the form $(\lambda x^{\mathcal{T}}.t)\alpha$. Typing constraints imply that in the first case the type for $\mu\alpha^{\mathcal{S}}.t$, of form $\mathcal{S} \Rightarrow \mathcal{T}$ had to be $\equiv_{fst}$-equivalent to a type $\mathcal{T}' \to \mathcal{T}''$, which implies

$\Pi(\_) : \Lambda\mu \longmapsto \lambda\mu\widehat{\mathsf{tp}}$	$\Sigma(\_) : \lambda\mu\widehat{\mathsf{tp}} \mapsto \Lambda\mu$
$\Pi(x) \triangleq x$ $\Pi(\lambda x.t) \triangleq \lambda x.\Pi(t)$ $\Pi((t)u) \triangleq (\Pi(t))\Pi(u)$ $\Pi(\mu\alpha.t) \triangleq \mu\alpha.[\widehat{\mathsf{tp}}]\Pi(t)$ $\Pi((t)\alpha) \triangleq \mu\widehat{\mathsf{tp}}.[\alpha]\Pi(t)$	$\Sigma(x) \triangleq x$ $\Sigma(\lambda x.t) \triangleq \lambda x.\Sigma(t)$ $\Sigma((t)u) \triangleq (\Sigma(t))\Sigma(u)$ $\Sigma(\mu\alpha.[\beta]t) \triangleq \mu\alpha.(\Sigma(t))\beta$ $\Sigma(\mu\alpha.[\widehat{\mathsf{tp}}]t) \triangleq \mu\alpha.(\Sigma(t))$ $\Sigma(\mu\widehat{\mathsf{tp}}.[\alpha]t) \triangleq (\Sigma(t))\alpha$ $\Sigma(\mu\widehat{\mathsf{tp}}.[\widehat{\mathsf{tp}}]t) \triangleq \Sigma(t)$

Fig. 6. Translations between $\Lambda\mu$-calculus and Herbelin & Ghilezan $\lambda\mu\widehat{\mathsf{tp}}$-calculus.

that $\mathcal{S}$ is of the form $\mathcal{T}' \to \mathcal{S}'$ and that $fst^{\to}$ could be applied. In the same way, in the second case, the typing constraints imply that subterm $\lambda x.t$, of type $\mathcal{T} \to \mathcal{T}'$ was $\equiv_{fst}$-equivalent to a type $\mathcal{S} \Rightarrow \mathcal{T}''$, which implies that $\mathcal{S}$ is $\equiv_{fst}$-equivalent to $\mathcal{T} \to \mathcal{S}'$ with $\mathcal{S}' \Rightarrow \mathcal{T}'' \equiv_{fst} \mathcal{T}'$ and then that $fst^{\to}$ could be applied.

The converse implications do not hold.

—There is a term t which is $fst^{\to}$-normal (e.g., because its μ-abstractions are all of an atomic type) and that is not fst-normal (no term containing a μ-abstraction is fst-normal).

—There is a term t which is fst^{-}-normal but not $fst^{\to}$-normal: $\lambda x^{A}.\mu\alpha^{B\to\mathcal{S}}.x$. □

4.4 Comments about the Type System $\Lambda_{\mathcal{S}}$

4.4.1 *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 [Pagani and Saurin 2008]: when translating $\Lambda_{\mathcal{S}}$ into (a kind of) polarized proof nets, $\mathcal{T}_1 \to \mathcal{T}_2$ becomes $?\mathcal{T}_1^{\perp} \parr \mathcal{T}_2$ as usual 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:

$$(?\mathcal{T}^{\perp} \parr \mathcal{S}) \parr \mathcal{T} \equiv_{fst} ?\mathcal{T}^{\perp} \parr (\mathcal{S} \parr \mathcal{T})$$

4.4.2 *Relation with $\lambda\mu\widehat{\mathsf{tp}}$ Type System.* Ariola et al. [2007] and Herbelin and Ghilezan [2008] introduced recently a calculus $\lambda\mu\widehat{\mathsf{tp}}$ that is a $\lambda\mu$-calculus with one dynamically bound variable, $\widehat{\mathsf{tp}}$ (see Figure 6):

$$\begin{array}{rcl} t, u & ::= & x \mid \lambda x.\, t \mid (t)u \mid \mu q.c \\ c & ::= & [q]t \\ q & ::= & \alpha \mid \widehat{\mathsf{tp}} \end{array}$$

This allows them to model call-by-value and call-by-name delimited continuations. 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 judgments: $\Gamma \vdash_{\Sigma} M : A; \Delta$ (Σ, annotating the $\vdash$ is a list of types).

The following connection holds between types in $\Lambda_{\mathcal{S}}$ and in $\lambda\mu\widehat{\mathsf{tp}}$: for a typing judgment of the form $x_1 : A^1_{\Sigma_1}, \ldots, x_k : A^k_{\Sigma_k} \vdash_{\Sigma} t \ : A | \alpha_1 : B_1, \ldots, \alpha_l : B_l$, term variables x_i of type $A^i_{\Sigma_i}$ will receive type $\tau(A^i \cdot \Sigma_i)$, term t will have type $\tau(A \cdot \Sigma)$ while stream variables α_i will have type $\sigma(B_i)$ where $\tau(\Sigma)$ and $\sigma(B)$ are defined as follows:

Definition 4.7 ($\tau(\Sigma)$, $\sigma(B)$).

$$\begin{array}{|ll|}\hline \tau(\bot) & = o \\ \tau(A \cdot \Sigma) & = \sigma(A) \Rightarrow \tau(\Sigma) \\ \sigma(o_i) & = o_i \to \bot \\ \sigma(A_{\Sigma} \to B) & = \tau(A \cdot \Sigma) \to \sigma(B) \\ \hline \end{array}$$

4.5 Properties of $\Lambda_{\mathcal{S}}$

4.5.1 *$\Lambda_{\mathcal{S}}$ 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 4.8. *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 Γ', Δ' and A' such that $\Gamma' \vdash_{\Lambda_{\mathcal{S}}} t \ : A' | \Delta'$.*

Definition 4.9. We consider a special term-type variable $o_{\bot}$ and we define the following transformations on the types of Parigot's $\lambda\mu$-calculus to $\Lambda_{\mathcal{S}}$ pre-types.

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

PROPOSITION 4.10. *Given a simple type A, then $A^{\mathcal{T}} \equiv_{fst} A^{\mathcal{S}} \Rightarrow o_{\bot}$.*

PROOF. The following equalities hold:

$$\begin{array}{lll} \text{(i)} & o^{\mathcal{S}} \Rightarrow o_{\bot} & = (o \to \bot) \Rightarrow o_{\bot} \\ & & = o^{\mathcal{T}} \\ \text{(ii)} & (A \to B)^{\mathcal{S}} \Rightarrow o_{\bot} & = (A^{\mathcal{T}} \to B^{\mathcal{S}}) \Rightarrow o_{\bot} \\ & & \equiv_{fst} A^{\mathcal{T}} \to (B^{\mathcal{S}} \Rightarrow o_{\bot}) \\ & & = A^{\mathcal{T}} \to B^{\mathcal{T}} \\ & & = (A \to B)^{\mathcal{T}} \end{array}$$ □

We now prove the theorem.

PROOF OF THEOREM 4.8. The result is proved by an induction on a $\lambda\mu$-typing derivation of term t.

A typing judgment of $\lambda\mu$-calculus $(x_i : A_i)_{i \in I} \vdash_{\lambda\mu} t \ : C | (\alpha_j : B_j)_{j \in J}$ is translated into a judgment of the form: $(x_i : {A_i}^{\mathcal{T}})_{i \in I} \vdash_{\Lambda_{\mathcal{S}}} t \ : C^{\mathcal{T}} | (\alpha_j : {B_j}^{\mathcal{S}})_{j \in J}$. We then have:

(1) $\dfrac{}{\Gamma, x : A \vdash_{\lambda\mu} x \ : A | \Delta}\ Var$ is turned to

$$\dfrac{\dfrac{\dfrac{\dfrac{}{y : A^{\mathcal{S}} \Rightarrow o_\perp \vdash y \ : A^{\mathcal{S}} \Rightarrow o_\perp | \alpha : A^{\mathcal{S}}, \beta : B^{\mathcal{S}}}\, Var_{\mathcal{T}}}{y : A^{\mathcal{S}} \Rightarrow o_\perp \vdash (y)\alpha \ : o_\perp | \alpha : A^{\mathcal{S}}, \beta : B^{\mathcal{S}}}\, App_{\mathcal{S}}}{y : A^{\mathcal{S}} \Rightarrow o_\perp \vdash \mu\beta.(y)\alpha \ : B^{\mathcal{S}} \Rightarrow o_\perp | \alpha : A^{\mathcal{S}}}\, Abs_{\mathcal{S}}}{\vdash \lambda y.\mu\beta.(y)\alpha \ : (A^{\mathcal{S}} \Rightarrow o_\perp) \rightarrow (B^{\mathcal{S}} \Rightarrow o_\perp) | \alpha : A^{\mathcal{S}}}\, Abs_{\mathcal{T}}}{x : ((A^{\mathcal{S}} \Rightarrow o_\perp) \rightarrow (B^{\mathcal{S}} \Rightarrow o_\perp)) \rightarrow (A^{\mathcal{S}} \Rightarrow o_\perp) \vdash (x)\lambda y.\mu\beta.(y)\alpha \ : A^{\mathcal{S}} \Rightarrow o_\perp | \alpha : A^{\mathcal{S}}}\, App_{\mathcal{T}}$$
$$\dfrac{x : ((A^{\mathcal{S}} \Rightarrow o_\perp) \rightarrow (B^{\mathcal{S}} \Rightarrow o_\perp)) \rightarrow (A^{\mathcal{S}} \Rightarrow o_\perp) \vdash ((x)\lambda y.\mu\beta.(y)\alpha)\alpha \ : o_\perp | \alpha : A^{\mathcal{S}}}{x : ((A^{\mathcal{S}} \Rightarrow o_\perp) \rightarrow (B^{\mathcal{S}} \Rightarrow o_\perp)) \rightarrow (A^{\mathcal{S}} \Rightarrow o_\perp) \vdash \mu\alpha.((x)\lambda y.\mu\beta.(y)\alpha)\alpha \ : A^{\mathcal{S}} \Rightarrow o_\perp |}\, Abs_{\mathcal{S}}$$
$$\dfrac{}{\vdash \texttt{call/cc} \ : (((A^{\mathcal{S}} \Rightarrow o_\perp) \rightarrow (B^{\mathcal{S}} \Rightarrow o_\perp)) \rightarrow (A^{\mathcal{S}} \Rightarrow o_\perp)) \rightarrow (A^{\mathcal{S}} \Rightarrow o_\perp) |}\, Abs_{\mathcal{T}}$$

with $\mathcal{T}_x = ((A^{\mathcal{S}} \Rightarrow o_\perp) \rightarrow (B^{\mathcal{S}} \Rightarrow o_\perp)) \rightarrow (A^{\mathcal{S}} \Rightarrow o_\perp)$.

Fig. 7. $\Lambda_{\mathcal{S}}$ type derivation for `call/cc`.

$$\dfrac{}{\Gamma^{\mathcal{T}}, x : A^{\mathcal{T}} \vdash_{\Lambda_{\mathcal{S}}} x \ : A^{\mathcal{T}} | \Delta^{\mathcal{S}}}\, Var_{\mathcal{T}}$$

(2) $$\dfrac{\Gamma, x : A \vdash_{\lambda\mu} t \ : B | \Delta}{\Gamma \vdash_{\lambda\mu} \lambda x^A.t \ : A \rightarrow B | \Delta}\, \lambda\text{-Abs} \quad \text{is turned to}$$
$$\dfrac{\Gamma^{\mathcal{T}}, x : A^{\mathcal{T}} \vdash_{\Lambda_{\mathcal{S}}} t \ : B^{\mathcal{T}} | \Delta^{\mathcal{S}}}{\Gamma^{\mathcal{T}} \vdash_{\Lambda_{\mathcal{S}}} \lambda x^{A^{\mathcal{T}}}.t \ : (A \rightarrow B)^{\mathcal{T}} | \Delta^{\mathcal{S}}}\, \lambda\text{-Abs}$$

(3) $$\dfrac{\Gamma \vdash_{\lambda\mu} t \ : A \rightarrow B | \Delta \quad \Gamma \vdash_{\lambda\mu} u \ : A | \Delta}{\Gamma \vdash_{\lambda\mu} (t)u \ : B | \Delta}\, \lambda\text{-App} \qquad \text{is turned to}$$
$$\dfrac{\Gamma^{\mathcal{T}} \vdash_{\Lambda_{\mathcal{S}}} t \ : (A \rightarrow B)^{\mathcal{T}} | \Delta^{\mathcal{S}} \quad \Gamma^{\mathcal{T}} \vdash_{\Lambda_{\mathcal{S}}} u \ : A^{\mathcal{T}} | \Delta^{\mathcal{S}}}{\Gamma^{\mathcal{T}} \vdash_{\Lambda_{\mathcal{S}}} (t)u \ : B^{\mathcal{T}} | \Delta^{\mathcal{S}}}\, \lambda\text{-App}$$

(4) $$\dfrac{\Gamma \vdash_{\lambda\mu} t \ : B | \Delta, \alpha : A}{\Gamma \vdash_{\lambda\mu} \mu\alpha^A(t)\beta \ : A | (\Delta, \beta : B) \setminus \alpha : A}\, \mu \qquad \text{is turned to}$$
$$\dfrac{\dfrac{\Gamma^{\mathcal{T}} \vdash_{\Lambda_{\mathcal{S}}} t \ : B^{\mathcal{T}} | \Delta^{\mathcal{S}}, \alpha : A^{\mathcal{S}}, \beta : B^{\mathcal{S}}}{\Gamma^{\mathcal{T}} \vdash_{\Lambda_{\mathcal{S}}} (t)\beta \ : o_\perp | \Delta^{\mathcal{S}}, \alpha : A^{\mathcal{S}}, \beta : B^{\mathcal{S}}}\, App_{\mathcal{S}}}{\Gamma^{\mathcal{T}} \vdash_{\Lambda_{\mathcal{S}}} \mu\alpha^{A^{\mathcal{S}}}.(t)\beta \ : A^{\mathcal{S}} \Rightarrow o_\perp | (\Delta^{\mathcal{S}}, \beta : B^{\mathcal{S}}) \setminus A^{\mathcal{S}}}\, Abs_{\mathcal{S}}$$

The previous proposition is sufficient to ensure that the derivation obtained by applying those rules is indeed a $\Lambda_{\mathcal{S}}$ typing derivation. □

Remark 4.11. 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 $\mathcal{S}_1 \Rightarrow (\mathcal{S}_2 \Rightarrow A)$.

4.5.2 *Typing* `call/cc` *in* $\Lambda_{\mathcal{S}}$. The $\Lambda\mu$-calculus encoding of `call/cc` is the term $\lambda x.\mu\alpha.((x)\lambda y.\mu\beta.(y)\alpha)\alpha$. In the classical type system presented in Figure 1, this term is typed by the Peirce's Law: $((A \rightarrow B) \rightarrow A) \rightarrow A$. In $\Lambda_{\mathcal{S}}$, `call/cc` can be assigned type $(((A^{\mathcal{S}} \Rightarrow o_\perp) \rightarrow (B^{\mathcal{S}} \Rightarrow o_\perp)) \rightarrow (A^{\mathcal{S}} \Rightarrow o_\perp)) \rightarrow (A^{\mathcal{S}} \Rightarrow o_\perp)$ as shown by the type derivation in Figure 7. One may notice that the structure of the Peirce's Law is now to be found in the stream type (see the alternation of $A^{\mathcal{S}}$ and $B^{\mathcal{S}}$ types).

4.5.3 *Subject Reduction.* In this paragraph, we prove that typed $\Lambda\mu$-calculus satisfies subject reduction. This property depends on three lemmas showing that the type is preserved by substitution when the substitution satisfies typing constraints.

LEMMA 4.12. *If* $\Gamma, x : A \vdash t \ : B|\Delta$ *and* $\Gamma \vdash u \ : C|\Delta$ *with* $A \equiv_{fst} C$*, then* $\Gamma \vdash t\{u/x\} \ : D|\Delta$ *with* $B \equiv_{fst} D$*.*

LEMMA 4.13. *If* $\Gamma \vdash t \ : A|\Delta, \alpha : \mathcal{S}, \beta : \mathcal{S}'$ *with* $\mathcal{S} \equiv_{fst} \mathcal{S}'$ *then* $\Gamma \vdash t\{\beta/\alpha\} \ : C|\Delta, \beta : \mathcal{S}'$ *with* $A \equiv_{fst} C$*.*

LEMMA 4.14. *If* $\Gamma \vdash t \ : A|\Delta, \alpha : B \to \mathcal{S}$*, then* $\Gamma, x : C \vdash t\{(u)x\beta/(u)\alpha\} \ : D|\Delta, \beta : \mathcal{S}'$ *with* $B \equiv_{fst} C$*,* $\mathcal{S} \equiv_{fst} \mathcal{S}'$ *and* $A \equiv_{fst} D$

PROOF. The lemmas are proved by a simple induction on the structure of term t on which the substitutions are applied. □

Typed $\Lambda\mu$-calculus satisfies subject reduction.

THEOREM 4.15 (SUBJECT REDUCTION). *Reduction of typed* $\Lambda\mu$*-terms preserves type: Let* $t, u \in \Lambda\mu$*. If* $\Gamma \vdash t \ : A|\Delta$ *and* $t \longrightarrow_{\Lambda\mu^{\to}} u$*, then* $\Gamma \vdash u \ : A|\Delta$*.*

PROOF. The proof is standard, by induction on the length of a derivation from t to u and by case on the first reduction rule which is applied. □

4.5.4 *Strong Normalization.* Finally, we prove strong normalization.

THEOREM 4.16 (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 [1997] for proving strong normalization of simply typed $\lambda\mu$-calculus. It consists in providing a translation of typed $\Lambda\mu$-terms 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 4.17. To each pre-type of $\Lambda_{\mathcal{S}}$, we associate a simple type as follows: We first enrich the set of type variables 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 pretypes 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 4.18. For each stream variable α, one considers new variables of λ-calculus: $\alpha_0, \alpha_1, \ldots, \alpha_n, \ldots$ The translation is then defined as follows:

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

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

PROPOSITION 4.19. *If t is a typed $\Lambda\mu$-term, then $\lceil t\rceil^{\Lambda_S}$ is a simply typed λ-term.*

PROOF. Easy by induction on the structure of a type derivation of t:

—$\equiv_{fst}$ steps disappear,
—$Abs_{\mathcal{T}}$ (respectively $App_{\mathcal{T}}$) is renamed $\to$-intro (respectively $\to$-elim) and
—$Abs_{\mathcal{S}}$ (respectively $App_{\mathcal{S}}$) is replaced by $n+1$ steps of $\to$-intro (respectively $\to$-elim) if n is the arity of the stream type. □

PROPOSITION 4.20 (SIMULATION). *Given two typed $\Lambda\mu$-terms t and u, the following facts hold:*

—*If $t \longrightarrow_{fst^{\to}} u$, then $\lceil t\rceil^{\Lambda_S} = \lceil u\rceil^{\Lambda_S}$*
—*If $t \longrightarrow_{\beta\eta} u$, then $\lceil t\rceil^{\Lambda_S} \longrightarrow^{+}_{\beta\eta} \lceil u\rceil^{\Lambda_S}$*

PROOF. Easy proof. □

Remark 4.21. One can actually be more precise and relate the number of β or η reductions needed to simulate a step of β_S or η_S to the arity of the type of the stream variable which is at stake in the reduction: it is $arity(\mathcal{S})+1$.

Remark 4.22. Note that the translation we chose for the $\mu\alpha.t$ and $(t)\alpha$ constructions were crucial in order to have simulation of β_S and η_S by the transitive and nonreflexive closure of $\beta\eta$ which will simplify the strong normalization proof in the following.

PROPOSITION 4.23 ($fst^{\to}$ TERMINATES). *Let t be a typed $\Lambda\mu$-term, there is no infinitely long $fst^{\to}$-derivation from t.*

PROOF. This is proved easily because at each $fst^{\to}$ reduction step, the arity of the type of some stream variable (the one to which the rule is applied) decreases. This ensures termination of $fst^{\to}$. □

PROPOSITION 4.24 ($\lceil\_\rceil^{\Lambda_S}$ IS REDUCTION-LENGTH INCREASING). *If $t \longrightarrow^{\star}_{\Lambda\mu^{\to}} u$ with m $\beta\eta$-reduction steps, then there exists a $\beta\eta$-reduction from $\lceil t\rceil^{\Lambda_S}$ to $\lceil u\rceil^{\Lambda_S}$ 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: $\delta = (t_i)_{i\geq 0}$ with $t = t_0$ and $t_i \longrightarrow_{\Lambda\mu^{\rightarrow}} t_{i+1}$ for every $i \geq 0$.

This reduction sequence contains only a finite number of $\boldsymbol{\beta\eta}$-reduction steps by Proposition 4.24: otherwise we would obtain an infinite $\beta\eta$-reduction sequence from $\lceil t \rceil^{\Lambda_S}$ in simply typed λ-calculus.

Thus, there is a integer n_0 such that all reduction in δ after $t_{n_0} \longrightarrow_{fst^{\rightarrow}} t_{n_0+1}$, are $fst^{\rightarrow}$-reduction. As a consequence, if δ were an infinite sequence there would be an infinite $fst^{\rightarrow}$-reduction sequence starting with t_{n_0}, which contradicts termination of $fst^{\rightarrow}$ which is ensured by Proposition 4.23.

Finally, we can conclude that typed $\Lambda\mu$-calculus is strongly normalizing. □

4.5.5 *$fst\boldsymbol{\eta}$-Long Forms.* We end this section by defining a notion of $fst\boldsymbol{\eta}$-long forms that are, for simply typed $\Lambda\mu$-calculus, the equivalent to η-long forms in simply typed λ-calculus.

Definition 4.25 (*$fst\boldsymbol{\eta}$-Long Form*). Let t a typed $\Lambda\mu$-term in canonical normal form of type $\mathcal{S}_1 \Rightarrow \mathcal{S}_2 \Rightarrow \ldots \mathcal{S}_m \Rightarrow A_1 \rightarrow \ldots A_n \rightarrow o$. One defines $[t]^{fst\eta\ell}$ as follows.

$$
\begin{array}{lcll}
[\lambda x^A.u]^{fst\eta\ell}_{abs} & = & \lambda x^A.[u]^{fst\eta\ell}_{abs} & \\
[\mu\alpha^{\mathcal{S}_1}.u]^{fst\eta\ell}_{abs} & = & \lambda x_1^{\alpha\, B_1} \ldots \lambda x_n^{\alpha\, B_n}.\mu\alpha^{\sigma}.[u]^{fst\eta\ell}_{abs} & \text{if } \mathcal{S}_1 = B_1 \rightarrow \ldots B_n \rightarrow \sigma \\
[t]^{fst\eta\ell}_{abs} & = & [\mu\alpha_1^{\mathcal{S}_1} \ldots \mu\alpha_m^{\mathcal{S}_m} \lambda x_1^{A_1} \ldots \lambda x_n^{A_n}. & \text{if } t = x, (u)v \text{ or } (u)\beta \text{ and} \\
 & & (t)\alpha_1 \ldots \alpha_m x_1 \ldots x_n]^{fst\eta\ell}_{abs} & \text{if } (m,n) \neq (0,0) \\
 & & & \\
[t]^{fst\eta\ell}_{abs} & = & [t]^{fst\eta\ell}_{app} & \text{if } m = n = 0 \text{ (\textit{ie.} } t \text{ of type } o) \\
[t]^{fst\eta\ell}_{app} & = & [t]^{fst\eta\ell}_{abs} & \text{if } t = \lambda x.u \text{ or } t = \mu\alpha.u \\
 & & & \\
[(u)v]^{fst\eta\ell}_{app} & = & ([u]^{fst\eta\ell}_{app})[v]^{fst\eta\ell}_{abs} & \\
[(u)\alpha]^{fst\eta\ell}_{app} & = & ([u]^{fst\eta\ell}_{app})[x_1^{\alpha}]^{fst\eta\ell}_{abs} \ldots [x_n^{\alpha}]^{fst\eta\ell}_{abs}\alpha & \text{if } u : (B_1 \rightarrow \ldots B_n \rightarrow \sigma) \Rightarrow \mathcal{T} \\
[x]^{fst\eta\ell}_{app} & = & x &
\end{array}
$$

5. CONCLUSION

One of the reasons for the success of Parigot's $\lambda\mu$-calculus as a proposal for studying control in functional programming is that it extends λ-calculus by keeping a large number of λ-calculus standard properties. However, it does not satisfy Böhm theorem. This drawback is fixed when moving to $\Lambda\mu$-calculus where separation property is recovered. This makes $\Lambda\mu$-calculus an interesting alternative to Parigot's original presentation, but $\Lambda\mu$-calculus meta-theory needed to be investigated.

The goal of this article was two-fold: to investigate the meta-theory of pure $\Lambda\mu$-calculus and to propose a new type system for $\Lambda\mu$-calculus in which some computational behaviors that are specific to $\Lambda\mu$-calculus (when compared to Parigot's calculus) can be observed.

Contributions of the Article. The main contributions of the article are as follows.

—We proved confluence of $\Lambda\mu$-calculus and obtained as a corollary a proof of the confluence for $\lambda\mu\eta$ which is simpler than the previously known proof by Py [1998].

—We provided characterizations of $\Lambda\mu$-calculus canonical normal forms strengthening the separation theorem for $\Lambda\mu$-calculus [Saurin 2005, 2008b].

—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 than Parigot's type system and in particular terms that were needed to obtain Böhm theorem in Saurin [2005], whereas they were not typable in the usual type system for $\lambda\mu$-calculi.

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 calculus $\lambda\mu\widehat{\mathsf{tp}}$ 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.

—An important motivation for providing $\Lambda\mu$ with a type system different from the original Parigot type system, was to investigate typed separation. Although there is no hope to obtain a typed separation result with a classical typing of $\Lambda\mu$, $\Lambda_{\mathcal{S}}$ stands as a serious candidate to validate typed separation theorem.

—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 $\Lambda_{\mathcal{S}}$ but this still needs to be investigated in more details.

—Second-order $\Lambda_{\mathcal{S}}$ shall be investigated.

—the interpretation of $\Lambda\mu$-calculus as a stream calculus was essential in designing $\Lambda_{\mathcal{S}}$. We wish to develop this aspect in two directions: (i) by studying the relationships between $\Lambda\mu$ and infinitary λ-calculi and (ii) by developing a true calculus of streams (some elements in this direction have been reported in Saurin [2008b]).

ACKNOWLEDGMENTS

The author wishes to thank Michele Pagani, Hugo Herbelin, and Silvia Ghilezan for helpful discussions and fruitful comments.

REFERENCES

ARIOLA, Z. M., HERBELIN, H., AND SABRY, A. 2007. A type-theoretic foundation of delimited continuations. *Higher-Order Symb. Comput*.

BABA, K., HIROKAWA, S., AND FUJITA, K.-E. 2001. Parallel reduction in type free lambda/mu-calculus. *Electron. Notes Theoret. Comput. Sci. 42*.

BÖHM, C. 1968. Alcune proprietà delle forme $\beta\eta$-normali nel λK-calcolo. *Publicazioni dell'Istituto per le Applicazioni del Calcolo* 696.

DAVID, R. AND PY, W. 2001. $\lambda\mu$-calculus and Böhm's theorem. *J. Symb. Logic*.

DE GROOTE, P. 1994. On the relation between the $\lambda\mu$-calculus and the syntactic theory of sequential control. In *Proceedings of the LPAR'94*. Lecture Notes in Articifial Intelligence, vol. 822. Springer-Verlag, Berlin.

DE GROOTE, P. 1998. An environment machine for the $\lambda\mu$-calculus. *Math. Struct. Comput. Sci. 8*.

DOŠEN, K. AND PETRIĆ, Z. 2000. The maximality of the typed lambda caluclus and of cartesian closed categories. *Publications de l'Institut Mathématique 68,* 82, 1–19. (Nouvelle série).

DOŠEN, K. AND PETRIĆ, Z. 2001. The typed Böhm theorem. In *Electron. Notes Theoret. Comput. Sci. 50,* 2.

FELLEISEN, M., FRIEDMAN, D. P., KOHLBECKER, E. E., AND DUBA, B. F. 1987. A syntactic theory of sequential control. *Theoret. Comput. Sci. 52*, 205–237.

FELLEISEN, M. AND HIEB, R. 1992. The revised report on the syntactic theories of sequential control and state. *Theoret. Comput. Sci. 103,* 2 (Sept.), 235–271.

GIRARD, J.-Y. 1991. A new constructive logic: classical logic. *Math. Struct. Comput. Sci. 1,* 3, 255–296.

GRIFFIN, T. 1990. A formulae-as-types notion of control. In *Proceedings of the Symposium on Principles of Programming Languages*. IEEE Computer Society Press, Los Alamitos, CA.

HERBELIN, H. AND GHILEZAN, S. 2008. An approach to call-by-name delimited continuations. In *Proceedings of the Symposium on Principles of Programming Languages*. ACM, New York.

HOWARD, W. A. 1980. The formulae-as-type notion of construction, 1969. In *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*, J. P. Seldin and R. Hindley, Eds. Academic Press, New York, 479–490.

JOLY, T. 2000. Codages, séparabilité et représentation de fonctions en λ-calcul simplement typé et dans d'autres systèmes de types. Ph.D. dissertation, Université Paris VII.

KRIVINE, J.-L. 1993. *Lambda-calculus, Types and Models*. Ellis Horwood.

MURTHY, C. R. 1992. A computational analysis of girard's translation and lc. In *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, Los, Alamitos, CA, 90–101.

ONG, L. 1996. A semantic view of classical proofs. In *Proceedings of the IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, CA.

PAGANI, M. AND SAURIN, A. 2008. Stream associative nets and $\Lambda\mu$-calculus. Tech. rep. 6431, INRIA.

PARIGOT, M. 1991. Free deduction: An analysis of "computations" in classical logic. In *Proceedings of the Russian Conference on Logic Programming*. Lecture Notes in Articifial Intelligence, vol. 592. Springer-Verlag, Berlin, 361–380.

PARIGOT, M. 1992. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In *Proceedings of the International Conference on Logic Programming and Automated Reasonning*. Lecture Notes in Computer Science, vol. 624. Springer-Verlag, Berlin.

PARIGOT, M. 1993. Strong normalization for second order classical natural deduction. In *Proceedings of the 8th Annual Symposium on Logic in Computer Science.* IEEE Computer Society Press, Los Alamitos, CA, 39–46.

PARIGOT, M. 1997. Proofs of strong normalisation for second order classical natural deduction. *J. Symb. Logic 62,* 4 (Dec.), 1461–1479.

PY, W. 1998. Confluence en $\lambda\mu$-calcul. Ph.D. dissertation, Université de Savoie.

SAURIN, A. 2005. Separation with streams in the $\Lambda\mu$-calculus. In *Proceedings of the IEEE Symposium on Logic In Computer Science*. IEEE Computer Society Press, Los Alamitos, CA, 356–365.

SAURIN, A. 2008a. On the relations between the syntactic theories of $\lambda\mu$-calculi. In *Proceedings of the 17th EACSL Annual Conference on Computer Science Logic 2008 (CSL 2008)*. Lecture Notes in Computer Science. Springer-Verlag, Berlin.

SAURIN, A. 2008b. Une étude logique du contrôle, appliquée à la programmation fonctionnelle et logique. Ph.D. dissertation, École Polytechnique.

STATMAN, R. 1983. λ-definable functionals and $\beta\eta$ conversion. *Archiv für Mathematische Logik und Grundlagenforschung 22*, 1–6.

Received October 2008; accepted February 2009