Separation with streams in the $\Lambda\mu$-calculus

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

Abstract

The $\lambda\mu$-calculus is an extension of the λ-calculus introduced in 1992 by Parigot [17] in order to generalize the Curry-Howard isomorphism to classical logic. Two versions of the calculus are usually considered in the literature: Parigot's original syntax and an alternative syntax introduced by de Groote. In 2001, David and Py [5] proved that the Separation Property (also referred to as Böhm theorem) fails for Parigot's $\lambda\mu$-calculus.

By analyzing David & Py's result, we exhibit an extension of Parigot's $\lambda\mu$-calculus, the $\Lambda\mu$-calculus, for which the Separation Property holds and which is built as an intermediate language between Parigot's and de Groote's $\lambda\mu$-calculi. We prove the theorem and describe how $\Lambda\mu$-calculus can be considered as a calculus of terms and streams. We then illustrate Separation in showing how in $\Lambda\mu$-calculus it is possible to separate the counter-example used by David & Py.

Keywords: Böhm Theorem, Calculus of Streams, Separation Property, Untyped $\lambda\mu$-calculus.

1. Introduction

The Separation Property. In 1968, Corrado Böhm [3] proved the Separation Property, an essential syntactical property of the pure λ-calculus which states that for any two $\beta\eta$-normal forms M and N of the λ-calculus that are syntactically different, there exists a context $C[]$ that separates them: the terms $C[M]$ and $C[N]$ will reduce to two different variables. This property has consequences both on the semantical and on the syntactical side. On the one hand it implies that a model of the λ-calculus cannot identify two different $\beta\eta$-normal forms without being trivial. On the other hand it means that there is a kind of adequation between the reduction rules and the syntactical constructions of the language: it is possible to explore normal terms completely by means of the computational rules. Separation is thus desirable since it is related to the question of whether the syntax and the reduction rules fit each other well.

The $\lambda\mu$-calculus. The $\lambda\mu$-calculus is an extension of the λ-calculus introduced in 1992 by Michel Parigot in order to generalize the Curry-Howard isomorphism to classical logic by developing a term assignment for a presentation of classical natural deduction.

The question of whether or not the $\lambda\mu$-calculus satisfies a kind of Separation Property has been addressed only recently (2001) and answered negatively by René David and Walter Py [5] by exhibiting a pair of "normal forms" that are not equivalent but non-separable.

In this paper we investigate the same question with a different point of view: we will wonder what is lacking in $\lambda\mu$-calculus for Separation to hold. As a result of a careful analysis of their work, we exhibit an extension of Parigot's $\lambda\mu$-calculus, the $\Lambda\mu$-calculus, for which the Separation Property actually holds. This is done by liberalizing the syntactical constructs of the calculus and by giving a new presentation of the $\lambda\mu$-calculus that stresses the interpretation of $\Lambda\mu$-calculus as a calculus of terms and streams of terms.

The extension obtained is not really new in the sense that it is an intermediate calculus between Parigot's and de Groote's presentations of $\lambda\mu$-calculus. This will actually help making clear the relationships between the two calculi.

Outline of the paper. In section 2, we briefly review the syntactical theory of the $\lambda\mu$-calculus: its origin and its basic definitions (the reduction rules and the Curry-Howard correspondence). The following section is dedicated to the study of David & Py's result, its analysis while the next section draws comments about non-separation and how to overcome its failure. The conclusions of sections 3 and 4 lead us naturally in section 5 to define an extension of $\lambda\mu$-calculus for which we can prove the Separation Theorem. This proof is outlined in section 6 following the spirit of Joly's proof technique for Separation in the λ-calculus which is a new, elegant and particularly short proof (much shorter than the original proofs of the result [3, 2, 13]) he gave in his PhD thesis [12]. As an illustration of our result, we show how in $\Lambda\mu$-calculus it is possible to separate the two terms of David & Py's counter-example to the Separation Property. Finally we analyze the question of typability of the separating contexts produced in our proof, discuss the interest of the streams and how this extension relates with different variants of the $\lambda\mu$-calculus (by Parigot and by de Groote).

Remark on notations. In the following, we will as usual consider the application to be left associative and we will consider two particular λ-terms (or $\lambda\mu$-terms or $\Lambda\mu$-terms according to the context): $1 = \lambda x, y.x$ and $0 = \lambda x, y.y$. Given an integer k, 1^k will denote the sequence of k terms all equal to 1: $(M)1^k$ corresponds to the term M applied to k arguments all equal to 1. We will also have to work with terms built from lots of applications: as a short for $(M)N_1 \dots N_k$ we will write either $(M)\vec{N_i}$ or $(M)\{N_i\}_{i \in I}$ depending on how precise we need our statement to be. Unless otherwise specified $\vec{N_i}$ (resp. I) may be empty.

2. $\lambda\mu$-calculus

Extending Curry-Howard to classical logic. Parigot's $\lambda\mu$-calculus [17] arose from aiming at extending the Curry-Howard isomorphism to classical logic by developing a term assignment for a presentation of minimal classical natural deduction. Several propositions preexisted (for instance Felleisen's $\lambda\mathcal{C}$-calculus [8] designed to provide a syntactic theory of control and later linked with classical logic by Griffin [11], or Girard's LC [9] which gives a presentation of classical logic based on the notion of polarities for which cut-elimination is confluent), but Parigot's approach differs from both of the previously mentioned theories. Indeed it is really a calculus (LC does not have a real syntax, it is still a logic), and the deductive power of classical logic is not recovered by adding an axiom ($\neg\neg A \Rightarrow A$ in the case of the $\lambda\mathcal{C}$-calculus) but by extending the logical rules (in defining a classical version of natural deduction). To each rule of the logic corresponds a syntactical construct on the computational side of the isomorphism, thus $\lambda\mu$-calculus has more syntactical components than the λ-calculus: μ-abstraction and naming of terms.

Since we want to analyze David & Py's result, our presentation of the $\lambda\mu$-calculus in this section closely follows the one given in their paper and we refer the reader to their paper for some definitions we won't recall here. Later in the paper we shall take some distance from this presentation.

Definition 1 (**$\lambda\mu$-terms**) *Let $\mathcal{V}$ (denoted by $x, y, \dots$) and $\mathcal{V}_c$ (denoted by $\alpha, \beta, \dots$) be two disjoint countable sets of term variables and of continuation variables respectively. $\lambda\mu$-terms are inductively defined by the following syntax:*
$M ::= x \mid \lambda x.M \mid \mu\alpha.[\beta]M \mid (M)M$
We call **named terms** *the expressions of the form $[\beta]M$.*

Two versions of the calculus are usually used in the literature: Parigot's original syntax and an alternative syntax ($M ::= x \mid \lambda x.M \mid (M)M \mid \mu\alpha.M \mid [\alpha]M$) which is used by several authors (de Groote [6, 7], Laurent [14], Ong and Stewart [16]...) even if Parigot's presentation seems to be a little more standard. Those two versions are generally considered as equivalent and their relation is never made really clear. Actually, on the logical point of view their relation is clear since one corresponds to minimal classical logic while the other is for classical logic with falsity, but computationally, their relation is not clear.

$$\frac{}{\Gamma, x : T \vdash x \ : T|\Delta}\ Var \qquad \frac{\Gamma, x : T \vdash M \ : U|\Delta}{\Gamma \vdash \lambda x.M \ : T \to U|\Delta}\ Abs$$

$$\frac{\Gamma \vdash M \ : T \to U|\Delta \quad \Gamma \vdash N \ : T|\Delta}{\Gamma \vdash (M)N \ : U|\Delta}\ App$$

$$\frac{\Gamma \vdash M \ : B|\Delta, \beta : B}{\Gamma \vdash \mu\alpha.[\beta]M \ : A|(\Delta \cup \{\beta : B\}) \setminus \{\alpha : A\}}\ \mu$$

Figure 1. Typing rules for $\lambda\mu$-calculus

$$\begin{array}{llll}
(\lambda x.M)N & \longrightarrow_\beta & M[N/x] & \\
\lambda x.(Mx) & \longrightarrow_\eta & M & x \notin FV(M) \\
[\beta]\mu\alpha.M & \longrightarrow_\rho & M[\beta/\alpha] & \\
\mu\alpha.[\alpha]M & \longrightarrow_\theta & M & \alpha \notin FV_c(M) \\
(\mu\alpha.M)N & \longrightarrow_\mu & \mu\alpha.M[[\alpha](U)N/[\alpha]U] & \\
\mu\alpha.M & \longrightarrow_\nu & \lambda x.\mu\alpha.M[[\alpha](U)x/[\alpha]U] & x \notin FV(M)
\end{array}$$

Figure 2. $\lambda\mu$-calculus reduction rules

We give in figure 1 the type system for the simply typed $\lambda\mu$-calculus. Erasing the proof terms, we get a presentation of minimal classical natural deduction. We do not enter into more details about the Curry-Howard isomorphism for $\lambda\mu$-calculus: this paper is mostly concerned with the untyped calculus. If interested, one can read Ariola & Herbelin's paper [1] which gives an interesting analysis of the links between minimal classical logic, $\lambda\mu$-calculus and Felleisen's $\lambda\mathcal{C}$.

$\lambda\mu$-calculus reduction rules. We present in figure 2 the reduction rules for $\lambda\mu$-calculus. We consider six reduction rules (β, η, μ, ν, ρ and θ) for $\lambda\mu$-calculus.
The substitution written $M[[\alpha](U)N/[\alpha]U]$ must be read as "all the subterms U of M named with α are replaced by $(U)N$". The substitution involved in the ρ-rule is just continuation variable renaming. Notice also that the ρ-reduction rule, as stated in figure 2, is not a proper rule of $\lambda\mu$-calculus since it does not involve terms of the calculus. To be precise, the ρ rule should be stated as: $\mu\alpha[\beta]\mu\gamma.[\delta]M \longrightarrow_\rho \mu\alpha.([\delta]M)[\beta/\gamma]$.

When designing the $\lambda\mu$-calculus, Parigot only considered the rules β, μ, ρ and θ. Having in mind the Separation Property, it is essential to have η-rule: without this rule the property is immediately false. Adding η to the four rules considered by Parigot is a bit problematic because it causes the failure of confluence (there is a critical pair μ–η for $\lambda x.(\mu\alpha.[\beta]M)x$, with $x \notin FV(M)$, see Py's PhD thesis [5, 18] for more details). Hence the need for introducing an additional rule called ν which corresponds to an η-expansion followed by a μ-reduction. Even in this case confluence holds only for terms with no free continuation variable (ie *μ-closed terms*). At this point the ν-rule already deserves some comments. First, note that it makes the μ-reduction superfluous since it can be simulated by a ν-reduction followed by a β-reduction. Second, note that with this rule we loose any hope of getting normal forms: as soon as a term contains a μ-abstraction it is possible to apply an arbitrarily large number of ν-rules to it.

3. The failure of Separation for the $\lambda\mu$-calculus

In the first paragraph of this section, we briefly summarize the content of the paper by David & Py proving the failure of Separation. The remaining of the section will be dedicated to the analysis of this result in order to make clear *why* and *how* separation fails in $\lambda\mu$-calculus.

Stating the failure of Separation. David & Py's result consists of two parts: they first develop tools to state the Separation Property in the $\lambda\mu$-calculus and then prove it is false.

To state Separation, a notion of **canonical normal forms** is introduced. They are the $\beta\eta\mu\rho\theta$-normal $\lambda\mu$-terms of the form $\lambda x_1 \dots x_n.((y)N_1 \dots N_k)$ or $\lambda x_1 \dots x_n.\mu\alpha[\beta]((y)N_1 \dots N_k)$ with $N_1, \dots, N_k$ in canonical normal form as well. These terms are not strictly speaking normal forms since a term containing a μ-abstraction has always an infinite computation because of the ν-rule. They are the $\beta\eta\mu\rho\theta$-normal forms for which no ν-reduction creates a redex for any of the five other reduction rules.

Given confluence and a notion of normal form, they can state the Separation Property for $\lambda\mu$-calculus that they show not to hold by exhibiting a counter-example, namely the $\lambda\mu$-term $W = \lambda x.\mu\alpha.[\alpha]((x)\ \mu\beta.[\alpha]((x)\ U_0\ y)\ U_0)$ with $U_0 = \mu\delta.[\alpha]0$.

Theorem 2 (Failure of Separation in $\lambda\mu$ [5]) *Let us consider $W_0 = W[0/y]$ and $W_1 = W[1/y]$.*
The Separation Property fails on W_0 and W_1: the terms are closed, in canonical normal form and are not $\beta\eta\mu\nu\rho\theta$-equivalent **but** *they are operationally equivalent*[1].

Analysis of the result. For the complete proof of the result, we refer the reader to David & Py's paper [5]. We now give briefly an intuitive idea of the reasons why W_0 and W_1 cannot be separated.

Thanks to a context lemma it is enough to consider only applicative contexts. As a consequence, it is sufficient to show that for no context $[]A\overrightarrow{B}$, $(W)A\overrightarrow{B}$ reduces to a head normal form for which y is the head variable. The reason for this can be summarized in the following derivation: $WA\overrightarrow{B} \rightarrow^{\star} \mu\alpha.[\alpha]((A)\ \mu\beta.[\alpha]((A)\ (\mu\delta.[\alpha](0)\ \overrightarrow{B})\ \mathbf{y}\ \overrightarrow{B})\ (\mu\delta.[\alpha](0)\ \overrightarrow{B})\ \overrightarrow{B})$

The first occurrence of A has then to select its first argument and move it in head position while the second occurrence of this term has to select its second argument. Up to this point, it is the exactly the same phenomenon as what happens with separation in the λ-calculus case. What differs from λ-calculus and makes the property fail here is that those two behaviours of the term A have to be achieved in the same context $\overrightarrow{B}$, which is impossible. The failure comes from this very point which is much different from the case of the λ-calculus where the arguments are consumed so that it would be possible to put the term A in two different contexts to make it compute differently.

4. Separation and Non-Separation

What can we expect now? Once the failure of Separation in $\lambda\mu$-calculus is proved, we can try to fix the problem and get the property back. The purpose of this section is to develop a general discussion about Separation and more precisely about ways to recover Separation when it fails.

Recovering Separation. The failure of Separation can be generally stated as follows: there exists two normal forms M and N that are **not equivalent** for the reduction rules but that **no context separates**. Let us illustrate these points by two very simple examples taken from the λ-calculus:

• Considering λ-calculus with only β as a reduction rule, Separation fails. If we add η, we get the property back: we have enlarged the equivalence class of the reduction rules and the result now becomes true. *The terms that were not separable are made equivalent.*

• Considering simply typed λ-calculus, Separation does not hold: we do not have enough contexts to explore all parts of the term. But if we remove the typing constraints, that is if we allow more contexts by building them with non-typable terms, one gets the Separation Property back. *The terms that were not equivalent are made separable.*

Those two directions suggest radically different methods to recover Separation. One way would be to try and find new reduction rules that would make more terms equivalent and in particular W_0 and W_1. The other direction would be to find a way to add new contexts among the possible

[1]In the usual sense that for all context $C[]$, $C[W_0]$ is solvable iff $C[W_1]$ is solvable, see [5] for complete definitions.

discriminating contexts in order to have a more powerful exploration of the terms and to satisfy Separation by this mean. In the following, we will take the second option and try to find out what contexts are missing in $\lambda\mu$-calculus.

Of course mixing both methods is also possible and an enlightening example of this can be found with the case of *designs* in Ludics [10]. Girard needed Separation to be satisfied for his para-proofs in order to use them as basic objects of an interactive theory of computation. The way he achieves Separation take aspects of both of the options mentioned above. From MALL proofs with a generalized axiom (✠), Girard first needs to define designs–dessins which are abstractions of MAAL proofs still built from sequents but sequents of addresses and with generalized logical rules. This first step would correspond to the enlargement of the class of contexts/tests available for Separation. Then he has to move to designs–desseins that are objects no more centered on sequents but on logical rules (which store the real information of proofs...) and which are kinds of equivalence classes of designs–dessins for an equivalence related with the structural rule of weakening. This second step is closer to the first option.

Typed Separation. Let us say a word about typed separation results before looking for the solution to the failure of separation for $\lambda\mu$-calculus in the untyped case.

Other versions of the Separation Property can be proved in the typed setting. Joly gave an intermediate result in his PhD thesis [12]: given two closed simply typed λ-terms t_1 and t_2 with same type that are not $\beta\eta$-equivalent, then by instantiating their type variables to the type of Church numerals $(o \to o) \to (o \to o)$ then one can find an applicative context in which the terms reduce to different Church numerals. These typed results are very different from the untyped Separation theorems we are interested in here and it would be the purpose of another paper to deal with separation for $\lambda\mu$-calculus in the typed case. Anyway we will discuss briefly this question in the last section of the paper.

Paradoxical consequence. A consequence of separation in λ-calculus that may seem surprising, not to say paradoxical, is that given two λ-terms (in normal form) M and B encoding different sorting algorithms (let us say a merge sort and a bubble sort for an encoding of lists of Church numerals), there will be a context in which these terms give different results even though they are correct encoding of the sorting algorithms. This is obtained by feeding the terms with some particular arguments which do not correspond to lists of Church numerals.
The typed version of the theorem tells us that, given these two terms, one can change their interface (we instantiate differently the type variables) and then it is possible to have them computing differently on well-typed arguments. This is not contradictory since after the type instantiation, the correct sorting functions have no reason to be still correct sorting functions (or to correspond to the same specification).

5. An extension of $\lambda\mu$-calculus: $\Lambda\mu$-calculus

In Section 3 we saw that the failure of Separation is due to the fact that the μ-abstraction consumes all its arguments and that it is not really possible to make a μ-abstraction disappear. Indeed the only rule that makes a μ disappear is the ρ rule, but in fact the term stays encapsulated inside a μ: $\mu\alpha[\beta]\mu\gamma.W \longrightarrow_\rho \mu\alpha.W[\beta/\gamma]$ (the θ rule is a little particular since it is an extensionality rule and thus the θ rule is kind of computationally transparent).
The reason for this has to be found in the fact that μ-abstraction and naming are a single syntactical construction in Parigot's $\lambda\mu$-calculus. In the current section, we get rid of this constraint in order to recover Separation.

$\Lambda\mu$: more liberal syntactical rules and a new syntax. We now extend the language by liberalizing the syntax in allowing a μ-abstraction on the one hand and a naming construct on the other hand, independent from each other. We will take the opportunity of this definition of a new language to change also the way naming is written: this simple change will have great impact on the writing of the definitions, the theorems (and their proofs) that will constitute the rest of the paper.

Definition 3 ($\Lambda\mu$-calculus) *Given two denumerable disjoint sets $\mathcal{V}_t$ (denoted by $x, y, z, \dots$) and $\mathcal{V}_s$ (denoted by $\alpha, \beta, \gamma, \dots$) of variables, the set of $\Lambda\mu$-terms is inductively defined as follows:* $M ::= x \mid \lambda x.M \mid \mu\alpha.M \mid (M)\alpha \mid (M)M$

We will refer to the elements of $\mathcal{V}_t$ and $\mathcal{V}_s$ as term variables (or λ-variables) and stream variables (or μ-variables) respectively. The terminology will be justified below.

Notice that we use the same notation for the application of a term and of a stream variable, this is for pure notational convenience and because it does not make anything ambiguous (since the elements of $\mathcal{V}_s$ are not terms of the language by themselves), but of course those two applications are of different kinds. The only "novelty" is the change of notation for naming and the use of the *fst*-rule which is a direct translation of the ν-rule defined by David & Py [5]. Moreover notice that we dropped the μ-rule since (as we mentioned previously) it is no more necessary and it has no particular meaning with the stream interpretation that we develop in the following.

$$\begin{array}{lll} (\lambda x.M)N & \longrightarrow_{\beta} M[N/x] & \\ \lambda x.(Mx) & \longrightarrow_{\eta} M & x \notin FV_t(M) \\ (\mu\alpha.M)\beta & \longrightarrow_{\beta_s} M[\beta/\alpha] & \\ \mu\alpha.(M\alpha) & \longrightarrow_{\eta_s} M & \alpha \notin FV_s(M) \\ \mu\alpha.M & \longrightarrow_{fst} \lambda x.\mu\alpha.M[(U)x\alpha/(U)\alpha] & x \notin FV_t(M) \end{array}$$

Figure 3. $\Lambda\mu$-calculus reduction rules

$$\frac{}{\Gamma, x : T \vdash x \ : T|\Delta}\, Var \qquad \frac{\Gamma, x : T \vdash M \ : U|\Delta}{\Gamma \vdash \lambda x.M \ : T \to U|\Delta}\, \lambda Abs$$
$$\frac{\Gamma \vdash M \ : T \to U|\Delta \quad \Gamma \vdash N \ : T|\Delta}{\Gamma \vdash (M)N \ : U|\Delta}\, \lambda App$$
$$\frac{\Gamma \vdash M \ : \bot|\Delta, \alpha : A}{\Gamma \vdash \mu\alpha.M \ : A|\Delta}\, \mu Abs \qquad \frac{\Gamma \vdash M \ : A|\Delta, \alpha : A}{\Gamma \vdash (M)\alpha \ : \bot|\Delta, \alpha : A}\, \mu App$$

Figure 4. Typing rules for $\Lambda\mu$-calculus

Definition 4 (Canonical normal forms for $\Lambda\mu$ (ΛCNF)) *The canonical normal forms are the $\beta\beta_s\eta\eta_s$-normal $\Lambda\mu$-terms of the form: $\lambda\overrightarrow{x_1}.\mu\alpha_1 \ldots \lambda\overrightarrow{x_k}.\mu\alpha_k.\lambda\overrightarrow{x_{k+1}}.(y)$ $\{M^i_1\}_{i\in I_1}\beta_1 \ldots \{M^i_l\}_{i\in I_l}\beta_l$ $\{M^i_{l+1}\}_{i\in I_{l+1}}$ with the M^j_i's in canonical normal form*[2].

Definition 5 (Head reduction, left reduction and *hnf*) *The head reduction is the reduction strategy defined as follows: apply head reduction for rules $\beta\eta\beta_s\eta_s$ as long as such a reduction step is possible. When there is no possible head reduction for $\beta\eta\beta_s\eta_s$ reduce (if it exists) the fst-redex which creates a head $\beta\eta\beta_s\eta_s$-redex (such a fst-redex is necessarily unique).*
The closed head normal forms (hnf) are the terms of the form: $\lambda\overrightarrow{x_1}.\mu\alpha_1 \ldots \lambda\overrightarrow{x_k}.\mu\alpha_k.\lambda\overrightarrow{x_{k+1}}.(y)$ $\{M^i_1\}_{i\in I_1}\beta_1 \ldots$ $\{M^i_l\}_{i\in I_l}\beta_l\{M^i_{l+1}\}_{i\in I_{l+1}}$ with arbitrary applications of stream variables and arbitrary terms M_i's (but such that no η nor η_s rule can be applied).
The left reduction strategy consists in applying the left-most possible $\beta\eta\beta_s\eta_s$ rule and to apply the fst-reduction only when it creates a $\beta\eta\beta_s\eta_s$-redex to the left of any other $\beta\eta\beta_s\eta_s$-redex. This reduction will be written $\to_l$.
When a term is not in hnf, left and head reduction coincide.

More terms, more space to η-expand. With the new language $\Lambda\mu$, we can build new terms (that have no equivalent in Parigot's $\lambda\mu$-calculus) such as $\mu\alpha.\mu\beta.(M)$ or $(M)\alpha x\beta$. Indeed, intuitively, η-expansion (not to mention θ-expansion) was forbidden between a μ-abstraction and a naming construction in $\lambda\mu$-calculus. With $\Lambda\mu$-calculus, we get the freedom to η-expand (with η or η_s) everywhere in the term: $\mu\alpha.\mu\beta.(M)\beta$ or $\mu\alpha.\lambda x.(M)x$ are now perfectly legal terms while neither $\mu\alpha.\mu\beta.[\beta]M$ nor $\mu\alpha.\lambda x.(M)x$ were allowed in $\lambda\mu$.

This change is extremely important for Separation: during the Separation process, η-expansion is critical (or to phrase it differently, the fact to saturate a term with arguments is essential to satisfy Separation).

Interpretation as a stream calculus. Let us now draw some comments about the links between $\Lambda\mu$-calculus and a calculus of streams in analyzing its reduction rules.

The reduction rules for $\Lambda\mu$-calculus are just the ones for $\lambda\mu$-calculus up to a change in the names (ρ becomes β_s, θ becomes η_s and ν becomes *fst*), the fact that μ disappears (as it is superfluous and has no particular meaning with the stream interpretation), and the fact that β_s is now a well-formed rule (ρ was not well formulated as mentioned when we defined the reduction rule for $\lambda\mu$-calculus).

Let us now give some intuitive interpretation of these rules. Clearly, η_s is an extensional rule while β_s has more computational content in a sense (even if it is a very weak form of β-reduction since it only involves variable renaming...). Moreover, the *fst*-rule relates term variables with stream variables and this is what makes the calculus interesting: it is a way to access the first term of the stream, to instantiate it. In order to make this point clear, let us now look at *fst*-derivations from a μ-abstracted term:

$$\mu\alpha.M \to^{\star}_{fst} \lambda x_1 \ldots \lambda x_n.\mu\alpha.M[(U)x_1 \ldots x_n\alpha/(U)\alpha]$$

Those derivations can be arbitrarily long so that the $\mu\alpha$ can be viewed as a kind of infinite λ-abstraction but not exactly, it is closer to an abstraction over streams of terms (or lazy stacks). This had already been noticed by Parigot in his original paper about $\lambda\mu$-calculus [17]: "The operator μ looks like a λ having potentially infinite number of arguments". This analogy is already valid with $\lambda\mu$-calculus but only takes its strength with $\Lambda\mu$-calculus in which Stream Abstraction and Stream Application are two well identified and totally independent constructions.

Definition 6 (Stream notation) *To shorten the writing of $\Lambda\mu$-terms, we shall adopt the following writing convention: when it is not ambiguous we abbreviate a sequence of abstractions of the form: $\lambda x_1 \ldots \lambda x_n.\mu\alpha.M$ as $\Lambda\mathcal{S}.M$ and a sequence of applications of the form $(M)M_1 \ldots M_m\alpha$ as $(M)\mathcal{S}$. For instance the canonical normal form of the previous definition will be written: $\Lambda\mathcal{S}_1 \ldots \Lambda\mathcal{S}_k.\lambda\overrightarrow{x_{k+1}}.(y)\mathcal{S}'_1 \ldots \mathcal{S}'_l\{M^i_{l+1}\}_{i\in I_{l+1}}$.*
In the rest of the paper, we refer to these notations as **Stream abstraction** *and* **Stream application** *respectively,*

[2]Notice that these are not all the $\beta\beta_s\eta\eta_s$-normal forms since for example $\mu\alpha.(\lambda x.x)\alpha$ is not in canonical normal form, but is in $\beta\beta_s\eta\eta_s$-normal form.
Note also that these canonical normal forms directly extend the ones for $\lambda\mu$-calculus: in $\lambda\mu$-calculus we always had $k = l = 0$ or $k = l = 1$ and the abstractions $\lambda\overrightarrow{x_2}$ as well as the sequence $(M^i_2)_{i\in I_2}$ are empty in this case.

and we will graphically denote them using a curly font for streams ($\mathcal{S}, \mathcal{M}, \mathcal{P}$...).

With this notation the stream interpretation becomes clearer and clearer, in particular it would be interesting to study reductions on streams like: $(\mu\alpha.M)\mathcal{S} \rightarrow M[\mathcal{S}/\alpha]$. This reduction would be more interesting than the pure variable renaming of the β_s rule and would look better than the strange substitution of the μ-reduction.

In addition, it is possible to make this notation even more uniform: the last layer of term applications in the canonical normal forms which is not very uniform can be encapsulated in a stream just by a η_s-expansion after all the abstractions: $\Lambda\mathcal{S}_1 \dots \Lambda\mathcal{S}_k.\lambda\overrightarrow{x_{k+1}}.(y)\mathcal{S}'_1 \dots \mathcal{S}'_l(M^i_{l+1})_{i \in I_{l+1}}$ $\rightarrow_{\eta_s exp} \Lambda\mathcal{S}_1 \dots \Lambda\mathcal{S}_{k+1}.(y)\mathcal{S}'_1 \dots \mathcal{S}'_{l+1}$

6. Recovering the Separation Property

6.1 Scheme of the proof for $\Lambda\mu$

Before going to the technical details of the proof of Separation, we informally outline the proof of the property.

Given two closed terms M and N that we want to separate, we must consider the open case and reason by induction on the *size* of the terms and by case on their structure.

The main case of the proof is for M, N of the form: $(x)\mathcal{M}_1 \dots \mathcal{M}_m M^1_{m+1} \dots M^k_{m+1}$ and $(y)\mathcal{N}_1 \dots \mathcal{N}_n N^1_{n+1} \dots N^l_{n+1}$.

When either $x \neq y$ or $(m, k) \neq (n, l)$, it is easy to separate the terms. Separation becomes more intricate in the case when $x = y$ and $m = n$ and $k = l$.

In this case, we need to find two subterms M' and N' at the same *position* in the trees of M and N that are not equivalent. In the λ-calculus case, it is evident but with $\Lambda\mu$, things are getting slightly more complicated. Indeed it could happen that the terms are of the form $(x)\alpha$ and $(x)y\beta$ (or even of the form $(x)\alpha$ and $(x)\beta$). In this case, there is no position satisfying the condition even if the terms are not equivalent. We should have done a *fst*-reduction on the $\mu\alpha$ and $\mu\beta$ when the terms were still closed and we would have got the terms $(x)v_\alpha\alpha$ and $(x)yv_\beta\beta$ for which there is no problem. This will be formalized in the Subterm Lemma (Lemma 18) which provides a way to prevent this problem from occurring. Then we select the subterms and finally separate them thanks to the induction hypothesis. This selection is achieved by assigning some term $\Phi^{\mathcal{P}}_{(i,j)}$ to the head variable x.

Nevertheless, it is not so easy to separate M and N since in addition to the selection of the subterms M' and N' by the term P assigned to the head variable x, we also need then to separate M' and N' and there is no reason for x not to appear also in the subterms M' and N' and thus P must be chosen to be compatible with the separation process of those terms which has still to be done.

In order to allow the term P assigned to x to play both roles (1. selection of the particular subterms M' and N' and 2. separation of $\Lambda\mu$-terms M' and N', respectively), we will need an auxiliary lemma which is the Parametric Pairs Lemma (Lemma 15). The aim of this lemma is to guarantee that it is possible to assign a more structured term to the variable x (a kind of pair of terms) without disturbing the Separation process.

For the terms that do not conform to the shape $(z)\mathcal{P}_1 \dots \mathcal{P}_m Q_1 \dots Q_k$, we need to reduce them to the previous case in such a way that it is compatible with the induction. To achieve that, the size that we shall define on the canonical normal forms of the $\Lambda\mu$-calculus must take into account this reduction in order not to have terms for which the size is not increasing during this step. The key point here is the stratification of the canonical normal forms in two layers: a layer of abstractions and a layer of applications (of streams and terms in both cases) and is proved in lemma 10.

6.2 Definitions and lemmas

In this section we introduce some structures, definitions and lemmas which are needed for the proof of the Separation theorem. Firstly we define the size of a $\Lambda\mu$-term to reason by induction.

Definition 7 (size of $\Lambda\mu$-terms) *We define inductively a size $\mathcal{T}$ on $\Lambda\mu$-calculus canonical normal forms:*
• $\mathcal{T}(x) = 0$
• $\mathcal{T}(\Lambda\mathcal{S}_1 \dots \Lambda\mathcal{S}_k.\lambda x_1 \dots \lambda x_l.M) = 1 + \mathcal{T}(M)$ *if M is not an abstraction and $k + l > 0$*
• $\mathcal{T}((x)\mathcal{S}_1 \dots \mathcal{S}_m N_1 \dots N_n) = 1 + \sum_{i=1}^m \mathcal{T}'(\mathcal{S}_i) + \sum_{i=1}^n \mathcal{T}(N_i)$ *if $(m, n) \neq (0, 0)$ where the size $\mathcal{T}'(\mathcal{S})$ of a stream application $\mathcal{S} = M_1 \dots M_m\alpha$ is $\sum_{i=1}^m \mathcal{T}(M_i)$*

Additionally to the size of a term we will need to talk about the subterms of certain terms (since separation is essentially an exploration of the terms). For this purpose, the size of a stream, positions of subterms and minimal applicative contexts will be important notions:

Definition 8 (Size of a Stream, Position of a subterm) *The size of a stream $\mathcal{S} = M_1 \dots M_n\alpha$ is defined as $\sharp\mathcal{S} = n$. Given a term M of the form $\Lambda\mathcal{S}_1 \dots \Lambda\mathcal{S}_k.\lambda x_1 \dots \lambda x_{k'}.(x)$ $\mathcal{S}'_1 \dots \mathcal{S}'_l M_1 \dots M_{l'}$ its set of positions $\mathcal{P}(M)$ is defined to be $\{(i, j), i \leq l, 1 \leq j \leq \sharp\mathcal{S}'_i\} \cup \{(l + 1, j), 1 \leq j \leq l'\}$ and the subterm of M at position $(i, j) \in \mathcal{P}(M)$ is the j^{th} element of $\mathcal{S}'_i$ if $i \leq l$ or is M_j if $i = l + 1$.*

Definition 9 (Minimal Applicative Context) *Given two $\Lambda\mu$-canonical normal forms M and N, a Minimal*

Applicative Context (MAC) is a context $\mathcal{C}$ of the form $[]x_{11}\dots x_{1k_1}\ \alpha_1\dots\alpha_n x_{(n+1)1}\dots x_{(n+1)k_{n+1}}$ of minimal length such that $\mathcal{C}[M]$ and $\mathcal{C}[N]$ reduce to ΛCNF of the form $(x)\mathcal{S}_1\dots\mathcal{S}_sP_1\dots P_p$ and such that the variables in $\mathcal{C}$ do not interfere with the free variables of M and N.

Lemma 10 *Consider two $\Lambda\mu$-terms M and N in canonical normal forms, $\mathcal{C}$ a MAC for this pair of terms and M' and N' reduced forms of $\mathcal{C}(M)$ and $\mathcal{C}(N)$ respectively.*
Then $\mathcal{T}(M')+\mathcal{T}(N')\leq\mathcal{T}(M)+\mathcal{T}(N)$.

Notice that in general the inequality is not strict even if the MAC in non-trivial. For example the terms $M=\mu\alpha.\lambda x.y$ and $N=z$ admit as a MAC the context $\mathcal{C}=[]\alpha x$ which leads to terms $M'=y$ and $N'=(z)\alpha x$ and the sum of the sizes has not decreased.

Definition 11 (Sub-term selector $\Phi^{\mathcal{P}}_{(i,j)}$) *The sub-term selector $\Phi^{\mathcal{P}}_{(i,j)}$ intended to select the sub-term at position (i,j) in a term $M=(x)\mathcal{S}_1\dots\mathcal{S}_sM_1\dots M_m$ is $\Phi^{\mathcal{P}}_{(i,j)}=\mu\alpha_1\dots\mu\alpha_{i-1}.\lambda x_{(i,1)}\dots\lambda x_{(i,j)}.\mu\alpha_i\dots\mu\alpha_s.\lambda x_{(s+1,1)}\dots\lambda x_{(s+1,m)}.x_{(i,j)}$ when (i,j) is in the set of positions $\mathcal{P}$ of the term M. (Of course, if $i=s+1$, then $\Phi^{\mathcal{P}}_{(i,j)}=\mu\alpha_1\dots\mu\alpha_s.\lambda x_{(s+1,1)}\dots\lambda x_{(s+1,m)}.x_{(s+1,j)}$).*

In the key part of the proof we shall need to replace a term by another one that carries more information: we need to store two programs respectively doing the work described in subsection 6.1. The usual way to do so in λ-calculus is to use pairs, but we need our information not to be retrieved immediately, we need the pair to accept several arguments before choosing what component it will restore. To this purpose a structure of parametric pairs is needed (it is simply a version with parameters of the usual pair encoding of λ-calculus):

Definition 12 (Parametric pairs: $\langle\_,\_\rangle_k$) *We define inductively $\langle U,V\rangle_k$ as follows:*
- $\langle U,V\rangle_0=\lambda z.(z)UV$ $\quad z\notin FV_t((U)V)$
- $\langle U,V\rangle_{k+1}=\mu\alpha.\langle(U)\alpha,(V)\alpha\rangle_k$ $\quad \alpha\notin FV_s((U)V)$

This may be rephrased as:
$\langle U,V\rangle_k=\mu\alpha_1\dots\mu\alpha_k.\lambda z.(z)((U)\alpha_1\dots\alpha_k)((V)\alpha_1\dots\alpha_k)$

Proposition 13 *The following relations hold:*
$\langle U,V\rangle_0 1\rightarrow^\star_l U$ *and* $\langle U,V\rangle_0 0\rightarrow^\star_l V$
$\langle U,V\rangle_{k+1}M\rightarrow^\star_l\langle(U)M,(V)M\rangle_{k+1}$ *and*
$\langle U,V\rangle_{k+1}\alpha\rightarrow^\star_l\langle(U)\alpha,(V)\alpha\rangle_k$
More globally, using the Stream Notation, we get:
$\langle U,V\rangle_k\mathcal{S}_1\dots\mathcal{S}_k1\rightarrow^\star_l(U)\mathcal{S}_1\dots\mathcal{S}_k$ *and*
$\langle U,V\rangle_k\mathcal{S}_1\dots\mathcal{S}_k0\rightarrow^\star_l(V)\mathcal{S}_1\dots\mathcal{S}_k$

At this point let us notice that the terms $\langle U,V\rangle_k$ are typical of the $\Lambda\mu$-calculus: we cannot build them in $\lambda\mu$-calculus as soon as $k\geq1$. This construction will have a key role in our proof. See section 7 for more details.

Notation 14 (The $\mathbb{1}^k$ Stream) *In the following, we write $\mathbb{1}^k$ for the stream $(1^k,\alpha)$. Such a stream will be used intensively.*

Lemma 15 (Parametric Pairs Lemma (PPL)) *Let τ,U be $\Lambda\mu$-terms and $x,y\in\mathcal{V}_t$ with $x\notin FV_t(U)$.*
If $\tau[U/x]\rightarrow^\star_l y$, then we have:
$\exists K\in\mathbb{N}$ *such that* $\forall k\geq K,\forall V\in\Lambda\mu$,
$\exists n^0$ *such that* $\forall(n_1,\dots,n_{k+1})$ *all greater or equal to* n^0,
$\exists(l_1,\dots,l_{k+1})\in\mathbb{N}^{k+1}$ *such that:*
$\tau[\langle U,V\rangle_k/x]\mathbb{1}^{n_1}\dots\mathbb{1}^{n_{k+1}}\rightarrow^\star_l(y)\mathbb{1}^{l_1}\dots\mathbb{1}^{l_{k+1}}$.

The lemma asserts that if $\tau[U/x]$ reduces to the variable y, then for any integer k big enough and for any term V, the parametric pair $\langle U,V\rangle_k$ will behave the same way as U does in the term τ as long as we feed it with enough streams $\mathbb{1}^n$ sufficiently large. The intuitive idea is that one can replace the term u by a term carrying more information but that still behaves the same way.

Definition 16 (Subterm Condition) *Two $\Lambda\mu$-terms M and N are said to satisfy the Subterm Condition if applied to a MAC, they reduce to $M'=(x)\mathcal{S}_1\dots\mathcal{S}_sM_1\dots M_m$ and $N'=(y)\mathcal{P}_1\dots\mathcal{P}_pN_1\dots N_n$ which are such that:*
- *$x\neq y$ or $s\neq p$ or $m\neq n$*
- *or there is a pair (i,j) both in $\mathcal{P}(M')$ and in $\mathcal{P}(N')$ such that the subterms of M' and N' at position (i,j) satisfy the Subterm Condition.*

Notice that the Subterm condition implies that the terms are not equivalent, the converse being false: usually non-equivalent terms in ΛCNF do not satisfy the Subterm condition as shown by the terms $(x)y\alpha$ and $(x)\alpha$. The ν-transformation deals with this.

Definition 17 (*fst*-transformation) *Given a closed $\Lambda\mu$-term M and a one-to-one map $\alpha\mapsto v_\alpha$ from $\mathcal{V}_s$ to $\mathcal{V}_t\setminus FV_t(M)$ we define $\lceil M\rceil$ as the result of the application of a fst-rule to each μ-abstraction of M:*
$\lceil x\rceil=x$
$\lceil\lambda x.M\rceil=\lambda x.\lceil M\rceil$ $\qquad \lceil(M)N\rceil=(\lceil M\rceil)\lceil N\rceil$
$\lceil(M)\alpha\rceil=(\lceil M\rceil)v_\alpha\alpha$ $\qquad \lceil\mu\alpha.M\rceil=\lambda v_\alpha.\mu\alpha.\lceil M\rceil$

Notice that in the previous definition $\lceil M\rceil$ actually does not depend on the choice of the function v since all the affected variables are bound.

Lemma 18 (Subterm Lemma) *Given two closed canonical normal forms that are not $\beta\eta\beta_s\eta_s fst$-equivalent, their fst-transforms satisfy the Subterm Condition.*

Proof sketch: The basic idea of the proof lies in the cases mentioned in section 6.1: $(x)M_1\dots M_m\alpha$ and $(x)N_1\dots N_n\beta$ with either $m\neq n$ or $\alpha\neq\beta$. Since the terms we consider are *fst*-transformed we have $M_m=v_\alpha$

and $N_n = v_\beta$ so that if $m \gneq n$ then $N_n = v_\beta \neq M_n$ and if $m = n$ then $N_n = v_\beta \neq v_\alpha = M_m$. Notice that in the lemma, the hypothesis of closed terms is only used to allow the *fst*-transformation of the terms. ∎

6.3 The main result

Theorem 19 (Separation Theorem for the $\Lambda\mu$-calculus) *Let M and N be closed $\Lambda\mu$-terms in canonical normal form which are not $\beta\eta\beta_s\eta_s fst$-equivalent. There exists an applicative context $C = []\mathcal{S}_1 \dots \mathcal{S}_k P_1 \dots P_l$ such that $C[M] \rightarrow_l^\star 1$ while $C[N] \rightarrow_l^\star 0$.*

Proof We will in fact separate the *fst*-transforms $\lceil M \rceil$ and $\lceil N \rceil$ of M and N respectively which will be easier since by lemma 18 they satisfy the Subterm Condition. It is easily checked that an applicative separating context for those terms is also a separating context for M and N.

We prove the result by induction on the sum of the sizes of the terms that we consider and by case on their structure. We reason on the open case on terms satisfying the Subterm Condition. The shape of the contexts we will build is: $\mathcal{C} = [] \left[\overrightarrow{U}/\overrightarrow{u}\right] \overrightarrow{\mathcal{S}}\,\overrightarrow{P}$.

First case: Both terms M and N are of the form $(x)\mathcal{P}_1 \dots \mathcal{P}_p Q_1 \dots Q_q$.
Let $M = (x)\mathcal{M}_1 \dots \mathcal{M}_m M_1 \dots M_{m'}$ and $N = (y)\mathcal{N}_1 \dots \mathcal{N}_n N_1 \dots N_{n'}$.

The only interesting case is when $x = y = s$ and $(m, m') = (n, n')$ (in the other cases, separation is direct by choosing an appropriate context). In that case, since the pair (M, N) satisfies the Subterm condition, we have a pair of integers (i, j) such that $M_{ij} \neq_{\beta\eta\beta_s\eta_s fst} N_{ij}$ and such that M_{ij} and N_{ij} satisfy the Subterm Condition. By induction hypothesis on (M_{ij}, N_{ij}), there are $\overrightarrow{U} = (U_u)_{u \in FV(M_{ij} N_{ij})}$ and $\overrightarrow{V} = \mathcal{V}_1 \dots \mathcal{V}_v V_1 \dots V_{v'}$ such that $M_{ij}\left[\overrightarrow{U}/\overrightarrow{u}\right]\overrightarrow{V} \rightarrow_l^\star 1$ and $N_{ij}\left[\overrightarrow{U}/\overrightarrow{u}\right]\overrightarrow{V} \rightarrow_l^\star 0$ so that for all variables z_1, z_2 we have $M_{ij}\left[\overrightarrow{U}/\overrightarrow{u}\right]\overrightarrow{V} z_1 z_2 \rightarrow_l^\star z_1$ and $N_{ij}\left[\overrightarrow{U}/\overrightarrow{u}\right]\overrightarrow{V} z_1 z_2 \rightarrow_l^\star z_2$.

If we write M' and N' for $M_{ij}\left[\overrightarrow{U}/\overrightarrow{u}, u \neq s\right]\overrightarrow{V} z_1 z_2$ and $N_{ij}\left[\overrightarrow{U}/\overrightarrow{u}, u \neq s\right]\overrightarrow{V} z_1 z_2$ and provided $z_1, z_2 \neq s$, we get $M'[U_s/s] \rightarrow_l^\star z_1$ and $N'[U_s/s] \rightarrow_l^\star z_2$.

At this point, we need to use the Parametric Pairs Lemma in order to pass two terms to the head variable s at once: on the one hand, we need a term whose role will be to select the subterms located in position (i, j) in M and N (that will be $\Phi_{(i,j)}^{\mathcal{P}(M)}$). and on the other hand a term whose role will be to separate the terms M_{ij} and N_{ij}.

By the PPL applied to M' and to N' with U_s and s, we get K such that for all $k \geq K$ and for all $(k_1, \dots, k_{\mathcal{Q}})$ big enough, there are $(l_1, \dots, l_{\mathcal{Q}})$ and $(l'_1, \dots, l'_{\mathcal{Q}})$ such that:
$M'[\langle U_s, \Phi \rangle_k / s]\mathbb{1}^{k_1} \dots \mathbb{1}^{k_{\mathcal{Q}}} \rightarrow_l^\star (z_1)\mathbb{1}^{l_1} \dots \mathbb{1}^{l_{\mathcal{Q}}}$
$N'[\langle U_s, \Phi \rangle_k / s]\mathbb{1}^{k_1} \dots \mathbb{1}^{k_{\mathcal{Q}}} \rightarrow_l^\star (z_2)\mathbb{1}^{l'_1} \dots \mathbb{1}^{l'_{\mathcal{Q}}}$
This can also be written (with U'_u equal to U_u except in the case of U'_s where it is equal to $\langle U_s, \Phi \rangle_k$):
$M_{ij}\left[\overrightarrow{U'}/\overrightarrow{x}\right]\overrightarrow{V} z_1 z_2 \mathbb{1}^{k_1} \dots \mathbb{1}^{k_{\mathcal{Q}}} \rightarrow_l^\star (z_1)\mathbb{1}^{l_1} \dots \mathbb{1}^{l_{\mathcal{Q}}}$ and
$N_{ij}\left[\overrightarrow{U'}/\overrightarrow{x}\right]\overrightarrow{V} z_1 z_2 \mathbb{1}^{k_1} \dots \mathbb{1}^{k_{\mathcal{Q}}} \rightarrow_l^\star (z_2)\mathbb{1}^{l'_1} \dots \mathbb{1}^{l'_{\mathcal{Q}}}$

Let us now define $w_1 = \mu\alpha_1 \dots \mu\alpha_{\mathcal{Q}}.1$ and $w_2 = \mu\alpha_1 \dots \mu\alpha_{\mathcal{Q}}.0$ and consider:

$$\begin{aligned}
& M\left[\overrightarrow{U'}/\overrightarrow{x}\right]\overrightarrow{V} w_1 w_2 \mathbb{1}^{k_1} \dots 0\mathbb{1}^{k_{k-m-v+1}} \dots \mathbb{1}^{k_{\mathcal{Q}}} \\
= \quad & \langle U_s, \Phi \rangle_k \mathcal{M}'_1 \dots \mathcal{M}'_m M'_1 \dots M'_{m'} \overrightarrow{V} \\
& w_1 w_2 \mathbb{1}^{k_1} \dots 0\mathbb{1}^{k_{k-m-v+1}} \dots \mathbb{1}^{k_{\mathcal{Q}}}
\end{aligned}$$

and the corresponding term for N.

We can now display a derivation from $\mathcal{C}(M)$ to 1 in this case: this is done in figure 5 (the left component of the pair is not shown since it is irrelevant for that part of the computation). The derivation from $\mathcal{C}(N)$ to 0 is similar.

$$\begin{aligned}
& M\left[\overrightarrow{U'}/\overrightarrow{x}\right]\overrightarrow{V} w_1 w_2 \mathbb{1}^{k_1} \dots \mathbb{1}^{k_{k-m-v}} 0\mathbb{1}^{k_{k-m-v+1}} \dots \mathbb{1}^{k_{\mathcal{Q}}} \\
& \rightarrow_l^\star \langle \star, (\Phi)\mathcal{M}'_1 \dots \mathcal{M}'_m {M'}_{m+1}^{1} \dots {M'}_{m+1}^{m'} \overrightarrow{V} \\
& \quad w_1 w_2 \mathbb{1}^{k_1} \dots \mathbb{1}^{k_{k-m-v}} \rangle_0 0\mathbb{1}^{k_{k-m-v+1}} \dots \mathbb{1}^{k_{\mathcal{Q}}} \\
& \rightarrow_l^\star \Phi \mathcal{M}'_1 \dots \mathcal{M}'_m {M'}_{m+1}^{1} \dots {M'}_{m+1}^{m'} \overrightarrow{V} w_1 w_2 \mathbb{1}^{k_1} \dots \mathbb{1}^{k_{\mathcal{Q}}} \\
& \rightarrow_l^\star (M'_{ij})\overrightarrow{V} w_1 w_2 \mathbb{1}^{k_1} \dots \mathbb{1}^{k_{k-m-v+1}} \dots \mathbb{1}^{k_{\mathcal{Q}}} \\
& \rightarrow_l^\star (w_1)\mathbb{1}^{l_1} \dots \mathbb{1}^{l_{\mathcal{Q}}} \quad \rightarrow_l^\star 1
\end{aligned}$$

Figure 5. Final derivation for theorem 19.

Second case: At least one of M and N is not of the form $(x)\mathcal{P}_1 \dots \mathcal{P}_p Q_1 \dots Q_q$.

By putting the terms in a Minimal Applicative Context, they reduce to terms M' and N' of the expected form that satisfy the Subterm Condition and $\mathcal{T}(M) + \mathcal{T}(N) \geq \mathcal{T}(M') + \mathcal{T}(N')$, so that it is possible to apply the induction hypothesis. ∎

6.4 Separating David & Py's counter-example

As an illustration of our result, we briefly exhibit a separating context for the terms W_0 and W_1 that induced failure of the Separation Property in David & Py's paper. We will show how it is possible to make the term $W = \lambda x.\mu\alpha.((x)\mu\gamma.((x)U_0 y\alpha)U_0\alpha)$ reduce to the variable y in a context $\mathcal{C}$, so that W_0 would reduce to 0 and W_1 to 1 in this context.

Writing $\mathcal{P}$ for the parametric pair: $\langle \lambda z_0, z_1.\mu\alpha.\mu\beta.z_1, \lambda x.\mu\alpha.x \rangle_1$, the separating context[3] is $\mathcal{C} = []\mathcal{P} x_0 x_1 \alpha 0 \alpha 1 \alpha$.

[3]This context is a simplified version of the context that can be extracted from the proof but it has the same shape. k_0 could be set to 0.

The separation process is described in the following derivation:

$$
\begin{array}{l}
\mathcal{C}(W) = (W)\mathcal{P}x_0x_1\alpha 0\alpha 1\alpha \\
\to_l^{\star} (\mathcal{P})\ \mu\gamma.((\mathcal{P})\ U_{x_1}yx_0x_1\alpha)\ U_{x_1}x_0x_1\alpha 0\alpha 1\alpha \\
\to_l^{\star} \langle \star, (\lambda x.\mu\alpha.x)\ \mu\gamma.((\mathcal{P})\ U_{x_1}yx_0x_1\alpha)\ U_{x_1}x_0x_1\alpha\rangle_0\ 0\alpha 1\alpha \\
\to_l^{\star} (\lambda x.\mu\alpha.x)\ \mu\gamma.((\mathcal{P})\ U_{x_1}yx_0x_1\alpha)\ U_{x_1}x_0x_1\alpha\alpha 1\alpha \\
\to_l^{\star} (\mu\gamma.(\mathcal{P})\ U_{x_1}yx_0x_1\alpha)\ \alpha 1\alpha \\
\to_l^{\star} (\mathcal{P})\ U_{x_1}yx_0x_1\alpha 1\alpha \\
\to_l^{\star} \langle(\lambda z_0, z_1.\mu\alpha.\mu\beta.z_1)\ U_{x_1}yx_0x_1\alpha, \star\rangle_0\ 1\alpha \\
\to_l^{\star} (\lambda z_0, z_1.\mu\alpha.\mu\beta.z_1)\ U_{x_1}yx_0x_1\alpha\alpha \\
\to_l^{\star} y
\end{array}
$$

7. Conclusion and future works: typability, streams and separation

Separation is an important property dealing with both syntax and semantics of languages. The way we managed to recover Separation witnesses the fact that the reductions were not responsible for the failure of Separation: the cause of this failure was that the syntax did not allow enough contexts to interact with the terms.

By changing only slightly the syntax we were able to get Separation back. This is a good illustration of how much Separation is sensitive to minor changes and that a right balance has to be found. Extending the syntax as we did was not necessarily leading to a solution to obtain Separation because Separation is non-monotonous: the more contexts we add in order to extend their separating power, the more terms we must be able to separate. On the other hand it results from the sensitivity of Separation as well as from its consequences that Separation can be considered as a major design criterion (or requirement) when defining a calculus as illustrated in Section 4 with Ludics. This idea of Separation as an important design principle was one of the ideas we followed when looking for Separation.

In the next paragraphs we give some elements of conclusions which are also future developments on which we are currently working or that we plan to study.

On Typability and Separation in $\Lambda\mu$-calculus. Although Separation is mainly a property of untyped calculi, typed versions of the property can be looked for as mentioned in Section 4 with Joly's typed result. In the same way, it could be of interest to look for typed separation results in $\Lambda\mu$-calculus. In this direction we formulate some remarks concerning typability in $\Lambda\mu$-calculus which are relevant with the problem of a typed result of Separation for $\Lambda\mu$-calculus.

We presented in figure 5 a type system for $\Lambda\mu$-calculus which is a standard $\lambda\mu$-calculus type system with $\perp$ [1, 14]. It is immediate to notice that every reduction rule of $\Lambda\mu$-calculus preserves the type except the *fst*-rule. Actually this rule does not even preserve typability as illustrated by the following example: $\lambda x.\mu\alpha.\mu\beta.(x\alpha) \to_{fst} \lambda x.\mu\alpha.\lambda y.\mu\beta.(x\alpha)$ The l.h.s.-term is typable of type $A \to A$ while the r.h.s.-term is not typable (for $\lambda y.\mu\beta.(x\alpha)$ would have to be of type $\perp$ which is not the case). This problem is not really a problem of the *fst*-rule (nor the ν-rule): such a phenomenon already occurs with the η expansion rule on which ν is built. In λ-calculus η_{exp} preserves typability but not types. As soon we introduce constant types to our type system, typability is no more preserved as examplified by $\lambda x.x+_{\mathtt{int}}x \to_{\eta_{exp}} \lambda x.\lambda y.(x+_{\mathtt{int}}x)y$.

This remark has important consequences on the question of typability of separation in $\Lambda\mu$-calculus: it is immediate to notice that the parametric pairs $\langle U, V\rangle_k$ of definition 12 are not typable as soon as $k > 0$. The proof we have for Separation is thus highly non-typable and it seems even stronger than that: Separation in $\Lambda\mu$-calculus (and thus in $\lambda\mu$-calculus) seems to rely on highly non typable constructions.

The Use of streams in $\Lambda\mu$-calculus. The originality of $\Lambda\mu$-calculus allowing us to prove Separation, namely the structure of streams, is thus essentially untypable and this is mainly what makes the difference between $\Lambda\mu$-calculus and the other presentations of $\lambda\mu$-calculus: as typed languages, they are essentially the same but $\Lambda\mu$-calculus has more possible untyped computations. One could consider that this difference between the typed computations and the untyped computations is a weakness of the calculus but on the contrary we think it is a richness of the calculus. Indeed our analysis lead us to a language in which new and interesting computational mechanisms can take place.

This presentation as a stream calculus is interesting for several reasons: first of all it gives a very clear computational interpretation to $\Lambda\mu$-calculus, in addition this structure is deeply rooted in the shape of $\Lambda\mu$-terms (and this fact is confirmed by the form the abstract machines for $\lambda\mu$-calculus [14, 4, 7] usually take) as mentioned when discussing the *fst*-rule. Moreover it makes rules nicer than the ones of the $\lambda\mu$-calculus especially suggesting a rule of the form $(\mu\alpha.M)\mathcal{S} \to M[\mathcal{S}/\alpha]$ and in this case the notion of substitution is much more uniform than the one of $\lambda\mu$-calculus.

Since the structure of streams seems to be of interest (at least because it lead to separation) it could be worth studying $\Lambda\mu$-calculus as a stream calculus with streams as first-class citizens (in the present paper we stayed close to the usual presentation of $\lambda\mu$-calculus introducing streams only as a convenient notation). Moreover leaving classical logic aside for a while it could even be interesting to develop a type system in which the streams can live and compute and for which typed separation results could be looked for as well as other standard results (SN, subject reduction...).

Relationships between the different presentations of $\lambda\mu$-calculus and $\Lambda\mu$-calculus. At the time of its design $\lambda\mu$-calculus was meant for the Curry-Howard isomorphism. It was essentially a typed language even in its untyped presentation since some typing constraints were directly built in the syntax of the language: in Parigot's presentation of $\lambda\mu$-calculus the fact that the μ-abstraction and naming are a single syntactical construction can be viewed as a typing constraint on the use of the $\perp$ type. On the other hand in de Groote's presentation of $\lambda\mu$-calculus which seems more liberal because it allows μ-abstraction and naming independently from each other (just like we did in $\Lambda\mu$-calculus), the typing constraint is reintroduced by another design of the calculus, the ϵ-rule, which is proper to de Groote's calculus (in addition to the ϵ-rule the formulation of the ρ-rule in de Groote's presentation also reintroduces such a constraint since, for confluence reasons, it forces the term to be in the form of Parigot's syntax). This rule is defined in the following way: $\mu\alpha.\mu\beta.M \to \mu\alpha.|M|_\beta$ (with $|M|_\beta$ is the term M where any free occurrence of the continuation variable β has been erased). We finally end up with two calculi, Parigot's and de Groote's calculi, which are equivalent except that de Groote's calculus has a little more terms available. On the opposite, $\Lambda\mu$-calculus really introduces new computations illustrated through the separation mechanism, this justifies to consider it as a different calculus from the traditional presentations of $\lambda\mu$-calculus.

Another Strategy to get Separation back in $\lambda\mu$-calculus? We highlighted two ways to recover Separation in $\lambda\mu$-calculus. We just showed that enlarging the class of separating contexts was successful; what about the other option? We have only partial results on this topic for the moment but we have a strong conjecture asserting the impossibility to recover separation in $\lambda\mu$-calculus by enlarging the equivalence class thanks to new reduction rules without equating almost all terms in the language. This would be a consequence of the non-locality of the problem of Separation on which the counter-example W has been built: basically the problem with the choice between first and second arguments can be hidden almost anywhere in the term.

Other future works. In addition to the four main directions already mentioned, we consider several other directions for future works. One direction would be to find a good denotational semantics for $\Lambda\mu$-calculus. The link of $\Lambda\mu$-calculus with Laurent's polarized proof-nets [15] has also to be explored further as well as the link of $\Lambda\mu$-calculus with the infinitary languages studied by Streicher [19] (for which separation fails with a counter-example having the same structure as W). The link with Streicher's work may actually be a good way to clarify questions about denotational semantics for $\Lambda\mu$-calculus.

Acknowledgments: I wish to thank Pierre-Louis Curien for his constant advice during this work. I also would like to thank Olivier Laurent, Thierry Joly and Dale Miller for valuable discussions regarding this work as well as the anonymous referees for useful comments on the organization and presentation of the paper.

References

[1] Z. Ariola and H. Herbelin. Minimal classical logic and control operators. In *ICALP'03*, volume 2719 of *LNCS*. Springer, 2003.

[2] H. Barendregt. *The Lambda Calculus, its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics. Elsevier publishers, 1984.

[3] C. Böhm. Alcune proprietà delle forme $\beta\eta$-normali nel λk-calcolo. *Publicazioni dell'Istituto per le Applicazioni del Calcolo*, 696, 1968.

[4] G. Bierman. A computational interpretation of the $\lambda\mu$-calculus. In *MFCS'98*, volume 1450 of *LNCS*, pages 336–345. Springer, 1998.

[5] R. David and W. Py. $\lambda\mu$-calculus and Böhm's theorem. *Journal of Symbolic Logic*, 66(1):407–413, 2001.

[6] P. de Groote. On the relation between the lambda-mu-calculus and the syntactic theory of sequential control. In *LPAR'94*, volume 822 of *LNAI*, pages 31–43, 1994.

[7] P. de Groote. An environment machine for the $\lambda\mu$-calculus. *MSCS*, 8:637–669, 1998.

[8] M. Felleisen and R. Hieb. The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 103(2):235–271, september 1992.

[9] J.-Y. Girard. A new constructive logic: classical logic. *MSCS*, 1(3):255–296, 1991.

[10] J.-Y. Girard. Locus solum. *MSCS*, 11:301–506, 2001.

[11] T. Griffin. A formulae-as-types notion of control. In *POPL'90*, pages 47–58, 1990.

[12] T. Joly. *Codages, séparabilité et représentation de fonctions en λ-calcul simplement typé et dans d'autres systèmes de types*. PhD thesis, Université Paris VII, 2000.

[13] J.-L. Krivine. *Lambda-calculus, Types and Models*. Ellis Horwood, 1993.

[14] O. Laurent. Krivine's abstract machine and the $\lambda\mu$-calculus (an overview). unpublished note, september 2003.

[15] O. Laurent. Polarized proof-nets and $\lambda\mu$-calculus. *Theoretical Computer Science*, 290(1):161–188, 2003.

[16] C.-H. Ong and C. A. Stewart. A Curry-Howard foundation for functional computation with control. In *POPL'97*, pages 215–227, 1997.

[17] M. Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In *International Conference on Logic Programming and Automated Deduction*, volume 624 of *LNCS*, pages 190–201. Springer, 1992.

[18] W. Py. *Confluence en $\lambda\mu$-calculus*. PhD thesis, Université de Savoie, 1998.

[19] T. Streicher. Laird domains. Manuscipt, 2003.