# Standardization and Böhm trees for $\Lambda\mu$-calculus

Alexis Saurin *

PPS & INRIA $\pi r^2$
alexis.saurin@pps.jussieu.fr

**Abstract.** $\Lambda\mu$-calculus is an extension of Parigot's $\lambda\mu$-calculus which (i) satisfies Separation theorem: it is Böhm-complete, (ii) corresponds to CBN delimited control and (iii) is provided with a stream interpretation. In the present paper, we study solvability and investigate Böhm trees for $\Lambda\mu$-calculus. Moreover, we make clear the connections between $\Lambda\mu$-calculus and infinitary $\lambda$-calculi. After establishing a standardization theorem for $\Lambda\mu$-calculus, we characterize sovalbility in $\Lambda\mu$-calculus. Then, we study infinite $\Lambda\mu$-Böhm trees, which are Böhm-like trees for $\Lambda\mu$-calculus; this allows to strengthen the separation results that we established previously for $\Lambda\mu$-calculus and to shed a new light on the failure of separation in Parigot's original $\lambda\mu$-calculus.
Our construction clarifies $\Lambda\mu$-calculus both as an infinitary calculus and as a core language for dealing with streams as primitive objects.

## 1 Introduction

**From $\lambda\mu$-calculus to $\Lambda\mu$-calculus.** Curry-Howard correspondence [14] originated as a correspondence between intuitionistic natural deduction (NJ) and simply typed $\lambda$-calculus. The extension of the correspondence to classical logic resulted in strong connections with control operators in functional languages as first noticed [12] by Griffin in his analysis of the logical interpretation of Felleisen's $\mathcal{C}$ operator [9]. Shortly after Griffin, Parigot introduced $\lambda\mu$-calculus [25] as an extension of $\lambda$-calculus corresponding to minimal classical natural deduction [24,1] in which one can encode usual control operators. $\lambda\mu$-calculus became one of the most widely studied classical $\lambda$-calculi, both in the typed and untyped setting. In particular, it extends $\lambda$-calculus (while retaining most of its standard properties) and intuitionistic natural deduction in a natural way.

However, a fundamental property of pure $\lambda$-calculus, known as separation property (or Böhm theorem [5]), does not hold for $\lambda\mu$-calculus [28,7]. In a previous work, we introduced $\Lambda\mu$-calculus, an extension to $\lambda\mu$-calculus, for which we proved that separation holds [30]. Since our result on separation theorem for $\Lambda\mu$-calculus, several authors investigated this calculus. While we studied confluence and type systems for $\Lambda\mu$-calculus [31,32,33], Lassen [20] introduced bisimulations

* Some elements of this works are presented in a fairly dense way due to lack of space. They shall be developed in an extended version of this paper which we are working on and which shall be made available at http://www.pps.jussieu.fr/~saurin/Publi/LMBT_long.pdf

for $\Lambda\mu$-calculus and introduced a CPS translation for $\Lambda\mu$-calculus and Herbelin and Ghilezan [13] showed that $\Lambda\mu$-calculus is a CBN calculus with delimited control.

**Delimited Control.** Herbelin and Ghilezan [13] evidenced that $\Lambda\mu$-calculus is indeed a call-by-name calculus with delimited continuations (in the spirit of Danvy and Filinski `shift`/`reset` operators [6]) using $\lambda\mu\widehat{\mathtt{tp}}$-calculus as a mediator between Danvy-Filinski CBV calculi and $\Lambda\mu$-calculus. Delimited control refers to a class of control operators which are much more expressive than non-delimited control operators (like `call/cc` for instance): they allow to simulate various side-effects [10], the monadic side-effects. In their seminal paper on `shift`/`reset` [6], Danvy and Filinski defined `shift`/`reset` operators by their CPS semantics. They also introduced a hierarchy of such control operators, $(\mathtt{shift}_i/\mathtt{reset}_i)_{i\in\omega}$, which are obtained by iterating CPS translations and that is known as the CPS hierarchy. Delimited control and the CPS hierarchy found applications in linguistics, normalization by evaluation, partial evaluation or concurrency. While the emphasis was traditionally given to the delimited-control languages in call-by-value, recent works [13,17] have advocated the study CBN delimited control. In a recent work [29], we introduced a hierarchy of calculi generalizing both $\lambda$-calculus and $\Lambda\mu$-calculus, *the Stream Hierarchy*, that we proved to be a call-by-name analogous to the CPS hierarchy.

**Streams and Infinitary $\lambda$-calculi.** Another viewpoint on the Separation property in $\Lambda\mu$-calculus is that the continuation variables in $\Lambda\mu$-calculus can be seen as abstracting streams of $\Lambda\mu$-terms. This provides the $\Lambda\mu$-calculus with an operational intuition of a stream calculus where one has the ability to abstract both on terms and streams. A weak form of this had already been noticed by Parigot who considered that "the operator $\mu$ looks like a $\lambda$ having potentially infinite number of arguments" [25]. The understanding of calculi of the family of $\lambda\mu$ as infinitary calculi is straightforward in $\Lambda\mu$-calculus: $\mu\alpha$ is considered as an abstraction over streams of terms (*ie.* $\lambda x_1^\alpha \dots x_n^\alpha \dots .t$) while $(t)\alpha$ can be seen as the application of a function $t$ to a stream of inputs (*ie.* $(t)x_1^\alpha \dots x_n^\alpha \dots$).

Infinitary $\lambda$-calculi have been considered in the literature [3,15,16,4,8] both to study infinite structures arising from lazy functional languages or to study consistency problems in the standard $\lambda$-calculus. Though, infinitary $\lambda$-calculi have been designed in a much different way from the infinitary calculus underlying $\Lambda\mu$-calculus: whereas in those frameworks, a reduction sequence may have transfinite length, terms have a (possibly infinite) depth which is bounded by $\omega$. On the contrary, the infinitary correspondent to $\mu\alpha.\mu\beta.\lambda x.x$ would be the transfinite term $\lambda x_0, x_1 \dots x_\omega, x_{\omega+1} \dots x_{\omega 2}.x_{\omega 2}$.

**Structure of the Paper.** The meta-theory of $\Lambda\mu$-calculus is already quite developed (Böhm and Church-Rosser theorems, simply typed calculus with strong normalization and subject reduction, complete CPS translations, abstract machines) as well as the connections with delimited control calculi. However, there

is no standardization theorem known for $\Lambda\mu$-calculus. In the same line of ideas, one may wonder how to construct Böhm trees for $\Lambda\mu$-calculus. The answers to these questions would help understanding the operational theory of $\Lambda\mu$-calculus as well as its model theory (by developing, for instance, a Böhm model for $\Lambda\mu$).

We shall precisely provide $\Lambda\mu$-calculus meta-theory with a standardization theorem and an analysis of Böhm trees in the present paper. In section 2, we shall recall some background on $\Lambda\mu$-calculus. We shall then define $\Lambda\mu$-head normal forms in Section 3, prove a standardization theorem and characterize solvability. In section 4, Böhm-like trees for $\Lambda\mu$-calculus will be introduced. A study of a Böhm tree semantics for $\Lambda\mu$-calculus is postponed to future work.

**Notations.** In the following, we shall use Krivine's notation [19] for $\lambda$-terms (as well as $\Lambda\mu$-terms or other $\lambda$-like calculi...): $\lambda$-application shall be written $(t)u$ instead of the notation $(MN)$ and we consider, as usual, $\lambda$-application to be left-associative, that is $(t)u_1 \dots u_{k-1}u_k$ shall be read as $(\dots((t)u_1)\dots u_{k-1})u_k$. Moreover we use an alternative notation for $\Lambda\mu$-terms, writing $(t)\alpha$ instead of the more common $[\alpha]t$ from Parigot (see [30,32] for explanations). For instance, we shall write $\mu\alpha.(t)u\beta$ for $\mu\alpha.((t)u)\beta$ (which would be written $\mu\alpha.[\beta](t\,u)$ in [25]).

## 2 Background and Notations on $\Lambda\mu$-calculus.

**Parigot's Original Calculus: $\lambda\mu$.** In 1992, Parigot introduced $\lambda\mu$, an extension of $\lambda$-calculus providing "an algorithmic interpretation of classical natural deduction" [25] by allowing for a proof-program correspondence *à la* Curry-Howard [14] between $\lambda\mu$-calculus and classical natural deduction [24,25]. Moreover, $\lambda\mu$ satisfies standard properties of $\lambda$-calculus such as confluence [25,28,7,2], subject reduction [25] and strong normalization [26,27]. However, there is no such thing as a Böhm theorem for $\lambda\mu$-calculus. Indeed, David & Py proved that separation fails in $\lambda\mu$ [28,7] by finding a counter-example (for details, see [7,30,32]).

**A $\lambda\mu$-calculus Satisfying Böhm Theorem: $\Lambda\mu$-calculus.** The failure of separation in $\lambda\mu$-calculus may be understood as the fact that some separating contexts are missing in $\lambda\mu$, making it impossible to *observe* (or access by Böhm out) the sub-parts of the terms that contain the difference. This led us to define, in a previous work [30], $\Lambda\mu$-calculus, an extension to $\lambda\mu$ 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 sufficient to realize a Böhm Out (in the new syntax).

**Definition 1.** *$\Lambda\mu$-terms ($t, u, v \dots \in \Sigma_{\Lambda\mu}$) are defined by the following syntax:*

$$\Sigma_{\Lambda\mu} \qquad t, u ::= \; x \mid \lambda x.t \mid (t)u \mid \mu\alpha.t \mid (t)\alpha$$

*where $x$ ranges over a set $\mathcal{V}_t$ of term variables and $\alpha$ ranges over a set $\mathcal{V}_s$ of stream variables. $\mathcal{V}_t$ and $\mathcal{V}_s$ are infinite and disjoint from each other. Closed $\Lambda\mu$-terms will be denoted by $\Sigma^c_{\Lambda\mu}$.*

**Remark 1** *Since $\alpha \notin \Sigma_{\Lambda\mu}$, it is clear that notations $(t)\alpha$ and $(t)u$ are not ambiguous.*

**Definition 2.** ***$\Lambda\mu$-reduction**, written $\longrightarrow_{\Lambda\mu}$, is induced by the following rules:*

$$\begin{array}{lll}
(\lambda x.t)u & \longrightarrow_{\beta_T} t\{u/x\} & \\
\lambda x.(t)x & \longrightarrow_{\eta_T} t & \text{if } x \notin FV(t) \\
(\mu\alpha.t)\beta & \longrightarrow_{\beta_S} t\{\beta/\alpha\} & \\
\mu\alpha.(t)\alpha & \longrightarrow_{\eta_S} t & \text{if } \alpha \notin FV(t) \\
\mu\alpha.t & \longrightarrow_{fst} \lambda x.\mu\alpha.t\{(v)x\alpha/(v)\alpha\} & \text{if } x \notin FV(t)
\end{array}$$

**Remark 2** *Notice that $\mu$-reduction (also called structural reduction, or $R_2$ in Parigot's papers, see [25]) is not part of $\Lambda\mu$-calculus reduction system. It can indeed be simulated by a fst-reduction followed by a $\beta_T$-reduction: $(\mu\alpha.t)u \longrightarrow_{fst} (\lambda x.\mu\alpha.t\{(v)x\alpha/(v)\alpha\})u \longrightarrow_{\beta_T} \mu\alpha.t\{(v)u\alpha/(v)\alpha\}$ and can be added for free.*

**Remark 3** *Names for reductions in $\Lambda\mu$-calculus come from the stream interpretation of $\Lambda\mu$: $\mu\alpha$ is seen as an* **abstraction over streams of terms** *while $(t)\alpha$ is a construction* **passing a stream as an argument** *to term $t$. In particular, $\mu$ cacn be viewed as an infinitary $\lambda$-abstraction. Under this interpretation, fst instantiates the* first *elements of a stream:*

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

**Definition 3 ($\boldsymbol{\beta}$, $\boldsymbol{\beta}^{var}$, $\boldsymbol{\eta}$).** *We consider the following subsystems of $\longrightarrow_{\Lambda\mu}$:*

- ***$\beta$** is the subsystem made of reductions $\beta_T$ and $\beta_S$;*
- ***$\eta$** is the subsystem made of reductions $\eta_T$ and $\eta_S$;*
- ***$\beta$fst** is the subsystem $\beta_T\beta_S$fst and **$\beta\eta$fst** for the full $\Lambda\mu$-reduction system;*
- *$\sim_{\Lambda\mu}$ is the equivalence associated with $\longrightarrow_{\Lambda\mu}$.*

The separation theorem for $\Lambda\mu$-calculus is stated with respect to a set of *canonical normal forms* (corresponding, in $\lambda$-calculus, to $\beta\eta$-normal forms):

**Definition 4.** *A $\Lambda\mu$-term $t$ is in* **canonical normal form (CNF)** *if it is **$\beta\eta$**-normal and if it contains no subterm of the form $(\lambda x.u)\alpha$ nor $(\mu\alpha.u)v$.*

*Remark 1.* A closed canonical normal form is thus a **$\beta\eta$**-normal form such that no *fst*-reduction creates a $\beta_T$-redex.

**Definition 5.** *$\Lambda\mu$-contexts are defined by the following syntax:*

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

**Theorem 4 (Böhm theorem for $\Lambda\mu$-calculus [30]).** *If $t$ and $t'$ are two non $\Lambda\mu$-equivalent closed canonical normal forms, there exists a context $\mathcal{C}$ such that:*

$$\mathcal{C}[t] \longrightarrow^{\star}_{\Lambda\mu} \lambda x.\lambda y.x \qquad \text{and} \qquad \mathcal{C}[t'] \longrightarrow^{\star}_{\Lambda\mu} \lambda x.\lambda y.y.$$

Confluence holds in $\Lambda\mu$-calculus [31,33] for $\mu$-closed terms:

**Theorem 5.** *For any $t, t', t'' \in \Sigma^c_{\Lambda\mu}$, there exists $u \in \Sigma^c_{\Lambda\mu}$ such that if $t \longrightarrow^\star_{\Lambda\mu} t', t''$ then $t', t'' \longrightarrow^\star_{\Lambda\mu} u$.*

**$\Lambda\mu$-calculus, a CBN calculus of delimited control.** Separation theorem for $\Lambda\mu$-calculus can be understood as the fact that $\Lambda\mu$-calculus admits more contexts than Parigot's original calculus allowing for a more powerful exploration of terms than in $\lambda\mu$-calculus. Typical contexts used in the separation proofs are $[]u_1 \dots u_m\beta_u v_1 \dots v_n\beta_v$. This exploits the fact that a context of the form $[]u_1 \dots u_m\beta_u$ *delimits* the part of the environment that can be passed through the left-most $\mu$-abstracted variable (*i.e.* $\alpha$) when term $\mu\alpha.\mu\alpha'.t$ is placed in the hole As a result, one can access to the second $\mu$-abstracted variable $\alpha'$ thanks to the second portion of the context, $v_1 \dots v_n\beta_v$. Based on this fact, Herbelin and Ghilezan [13] evidenced strong connections between $\Lambda\mu$-calculus and calculi with delimited continuations in the spirit of Danvy and Filinski `shift`/`reset` operators [6] using the calculus $\lambda\mu\widehat{\mathsf{tp}}$ (where $\widehat{\mathsf{tp}}$ is dynamically bound, see [13]):

| $\Sigma_{\lambda\mu\widehat{\mathsf{tp}}}$ | $t, u ::= x \mid \lambda x.t \mid (t)u \mid \mu q.c$ | $c ::= [q]t$ | $q ::= \alpha \mid \widehat{\mathsf{tp}}$ |
|---|---|---|---|

In its call-by-value version, $\lambda\mu\widehat{\mathsf{tp}}$ is equivalent to Danvy-Filinski's `shift`/`reset` operators while in its call-by-name version the calculus is equationally correspondent to $\Lambda\mu$. This led Herbelin & Ghilezan to assert that $\Lambda\mu$ is a CBN calculus of delimited control, providing an additional evidence of the striking difference between $\lambda\mu$ and $\Lambda\mu$ due to the slight change of syntax operated in [30].

Delimited control operators are much more expressive than non-delimited control operators (like `call/cc` for instance) in that they allow to simulate various side-effects [10]. Delimited control found several applications in linguistics, normalization by evaluation, partial evaluation or concurrency.

## 3 Standardization Theorem for $\Lambda\mu$-calculus

In this section, we shall prove a standardization theorem for $\Lambda\mu$-calculus.

### 3.1 $\Lambda\mu$-Head Normal Forms.

**Definition 6 (Pre-redex).** *Let $v \in \Sigma_{\Lambda\mu}$ be a closed term. A subterm $v'$ of $v$ is a* **pre-redex** *(or p.r.) if it is of one of the following forms: $(\lambda x.t)u$, $(\lambda x.t)\alpha$, $(\mu\alpha.t)u$ or $(\mu\alpha.t)\beta$. The four types of pre-redex are respectively denoted as $(\mathcal{T})\mathcal{T}$, $(\mathcal{T})\mathcal{S}$, $(\mathcal{S})\mathcal{T}$ and $(\mathcal{S})\mathcal{S}$ pre-redexes.*

**Lemma 1.** *Considering **$\beta$fst**, a p.r. of $t \in \Sigma_{\Lambda\mu}$ can be reduced in different ways:*

- *a $(\mathcal{T})\mathcal{T}$ p.r. is a $\beta_T$-redex: it can be reduced by exactly one instance of $\beta_T$;*

– *a $(\mathcal{S})\mathcal{T}$ p.r. is not a redex, but can be reduced to a $(\mathcal{T})\mathcal{T}$ p.r. (which is a $\beta_T$-redex) thanks to an instance of fst (there is exactly one instance of fst-reduction which realizes this);*
– *a $(\mathcal{T})\mathcal{S}$ p.r., $(\lambda x.t)\alpha$, can be treated as above, provided $\alpha$ is bound in $t$;*
– *a $(\mathcal{S})\mathcal{S}$ p.r. is a $\beta_S$-redex and can thus be reduced thanks to an instance of $\beta_S$. Moreover, and contrarily to any other type of pre-redex, two other rules can affect this p.r.: suppose that the p.r. is of shape $(\mu\alpha.u)\beta$. Then one can also apply a fst-reduction on $\mu\alpha$ creating a $(\mathcal{T})\mathcal{S}$ p.r., or apply a fst-reduction to the $\mu$-abstraction which binds variable $\beta$ and create a $(\mathcal{S})\mathcal{T}$ p.r.*

**Definition 7.** – *$t$ is an* **application term** *if it is of form $(u)v$ or $(u)\alpha$;*
– *$t$ is an* **abstraction term** *if it is of form $\lambda x.u$ or $\mu\alpha.u$.*

A $\Lambda\mu$-term $t$ is either a variable, or an application term, or an abstraction term. An abstraction term $t$ is of the form $\lambda\overrightarrow{x_0}.\mu\alpha_0.\ldots.\lambda\overrightarrow{x_n}.\mu\alpha_n.\lambda\overrightarrow{x_{n+1}}.t_0$ where $t_0$ is not an abstraction term. An application term $t$ is of the form $(t_0)\overrightarrow{t_1}\alpha_1\ldots\overrightarrow{t_m}\alpha_m\overrightarrow{t_{m+1}}$ where $t_0$ is not an application term.

**Lemma 2.** *Any $\Lambda\mu$-term $t$ has the following form:*

$$\lambda\overrightarrow{x_0}.\mu\alpha_0.\ldots.\lambda\overrightarrow{x_n}.\mu\alpha_n.\lambda\overrightarrow{x_{n+1}}.(t_0)\overrightarrow{t_1}\beta_1\ldots\overrightarrow{t_m}\beta_m\overrightarrow{t_{m+1}} \qquad (\star)$$

*where $t_0$ is either (case a) a variable $x$ or (case b) a pre-redex of $t$.*

**Definition 8 (Head normal forms).**

– *$t \in \Sigma^c_{\Lambda\mu}$ is in* **head normal form** *(hnf) if $t_0$ is a variable $x$ in representation $(\star)$; in this case $x$ is the* **head variable** *of $t$;*
– *otherwise, $t_0$ is a pre-redex and is called the* **head pre-redex** *(or hpr) of $t$.*
– *$\Lambda\mu$-HNF is the set of head normal forms;*
– *$t$ has a hnf if $t \longrightarrow^\star_{\Lambda\mu} t'$ with $t'$ a hnf;*
– *head normal forms are also the closed terms given by the following grammar:*

$$h ::= g \mid \lambda x.h \mid \mu\alpha.h \qquad g ::= x \mid (g)t \mid (g)\alpha$$

**Definition 9 (Head reduction). Head reduction**, *denoted $\longrightarrow_h$, is the sub-reduction of **$\beta$fst** which reduces the hpr if there is one. A* **head reduction path** *(or hrp) for a term $t$ is a sequence $t_0, t_1, \ldots$ such that $t = t_0 \longrightarrow_h t_1 \longrightarrow_h \ldots$*

*If $t_n$ is in hnf for some $n$, then the hrp for $t$ terminates in $t_n$, otherwise $t$ has an infinite head reduction.*

Notice that, contrarily to $\lambda\beta$-calculus, the hrp of $t$ is not necessarily unique.

*Example 1.* For instance if $t = \mu\alpha.(\mu\beta.x)\alpha$ and $u = \lambda y.\mu\alpha.x$, the following two reduction sequences are head reduction paths from $t$ to $u$:

$$\begin{array}{ll} t = & \mu\alpha.(\mu\beta.x)\alpha \\ \longrightarrow_{fst} & \lambda y.\mu\alpha.(\mu\beta.x)y\alpha \\ \longrightarrow_{fst} & \lambda y.\mu\alpha.(\lambda z.\mu\beta.x)y\alpha \\ \longrightarrow_{\beta_T} & \lambda y.\mu\alpha.(\mu\beta.x)\alpha \\ \longrightarrow_{\beta_S} & \lambda y.\mu\alpha.x \end{array} \qquad \begin{array}{ll} t = & \mu\alpha.(\mu\beta.x)\alpha \\ \longrightarrow_{fst} & \mu\alpha.(\lambda z.\mu\beta.x)\alpha \\ \longrightarrow_{fst} & \lambda y.\mu\alpha.(\lambda z.\mu\beta.x)y\alpha \\ \longrightarrow_{\beta_T} & \lambda y.\mu\alpha.(\mu\beta.x)\alpha \\ \longrightarrow_{\beta_S} & \lambda y.\mu\alpha.x \end{array}$$

### 3.2 Standard reductions

We shall now define standard reductions for $\Lambda\mu$-calculus and prove standardization. The notion of standard reduction is more complex than in $\Lambda\mu$-calculus because of the structure of *fst*-reduction. Indeed, *fst*-reduction is a non-local rule which acts both at the place of the $\mu$-abstracted sub-term $\mu\alpha.t$ but also at the occurrences of abstracted variable: $(t)\alpha$. This is reminiscent of a proof-net analysis of $\Lambda\mu$-calculus pursued together with Pagani [23] in which the proof-net counter-part of *fst*-reduction could be activated either at the level of the abstraction or at the level of the variables.

In our analysis of standardization, the appropriate notion will thus be that of p.r.: to each $\boldsymbol{\beta}$**fst**-reduction on a given $\Lambda\mu$-term $t$, we shall associate *at most* one p.r. That p.r. will be used to determine whether a given reduction step extends a standard reduction sequence into a larger reduction sequence.

**Definition 10.** *Let $t, u \in \Sigma_{\Lambda\mu}$ be closed terms such that $t \xrightarrow{\rho}_{\boldsymbol{\beta fst}} u$. Depending on the reduction step, $\rho$, which is applied and the structure of $t$, we shall associate zero or one p.r. of $t$ which will be called the* **pre-redex associated with** *$(t, \rho)$.*

- *Suppose $t \xrightarrow{\rho}_{\beta_T} u$. There exists a unique pair $(\mathcal{C}_\rho, t_\rho)$ such that $t = \mathcal{C}_\rho[t_\rho]$, $t_\rho = (\lambda x.v)w$ and $u = \mathcal{C}_\rho[v\{w/x\}]$. Then, $t_\rho$ is the p.r. associated with $(t, \rho)$.*
- *Suppose $t \xrightarrow{\rho}_{\beta_S} u$. There exists a unique pair $(\mathcal{C}_\rho, t_\rho)$ such that $t = \mathcal{C}_\rho[t_\rho]$, $t_\rho = (\mu\alpha.v)\beta$ and $u = \mathcal{C}_\rho[v\{\beta/\alpha\}]$. Then , $t_\rho$ is the p.r. associated with $(t, \rho)$.*
- *Finally, suppose $t \xrightarrow{\rho}_{fst} u$. There exists a unique pair $(\mathcal{C}_\rho, t_\rho)$ such that $t = \mathcal{C}_\rho[t_\rho]$, $t_\rho = \mu\alpha.v$ and $u = \mathcal{C}_\rho[\lambda x.\mu\beta.v\{(w)x\beta/(w)\alpha\}]$. We consider three cases:*
  - *either $\mathcal{C}_\rho = \mathcal{C}'_\rho[([])w]$. In this case, then $(t_\rho)w$ is a p.r. and is the* **pre-redex associated with** *$(t, \rho)$ and $\rho$ is an* **essential** *fst-reduction step;*
  - *otherwise, if there is a subterm $r$ of $v$ which is a p.r. of the form $(\lambda x.w)\alpha$ or $(\mu\beta.w)\alpha$ where $\alpha$ is free in $v$, then the* **pre-redex associated with** *$(t, \rho)$ is the left-most such $r$ and $\rho$ an* **essential** *fst-reduction step;*
  - *otherwise, there is no such $(\lambda x.w)\alpha$ nor $(\mu\beta.w)\alpha$ (i.e. either $\alpha \notin FV(v)$ or all free occurrences of $\alpha$ are of the form $(x)\alpha, ((w')w'')\alpha$ or $((w')\beta)\alpha$). Then $\rho$ is said* **superfluous** *and no p.r. is associated with $(t, \rho)$.*

  *We write $fst^e$ to denote an essential fst-reduction step and $fst^s$ to denote a superfluous fst-reduction step.*

**Definition 11 (Active symbol).** *A symbol $\lambda$ or $\mu$ is* **active** *if it is the first symbol of a pre-redex.*

**Definition 12 (Degree).** *Let $t \in \Sigma^c_{\Lambda\mu}$ and $r$ be a p.r. of $t$. The* **degree** *$d(r)$ of $r$ is the number of $\lambda$ or $\mu$ which are both active in $t$ and to the left of $r$ in $t$.*

*The* **degree of a reduction step** *$\rho$, $d(\rho)$ is defined as follows:*

- *if $\rho$ is a $\boldsymbol{\beta}$-reduction or a $fst^e$-reduction, then $d(\rho)$ is the degree of the pre-redex associated with $(t, \rho)$;*
- *if $\rho$ is a $fst^s$-reduction and $n$ is the number of $\mu$-abstraction to the left of the $\mu$-abstraction corresponding to $\rho$, then $d(\rho) = \omega + n$.*

**Definition 13 (Standard reduction).** *A (possibly infinite) reduction sequence* $t = t_0 \xrightarrow{\rho_1} t_1 \xrightarrow{\rho_2} \dots$ *is* **standard** *if* $d(\rho_i) \leq d(\rho_j)$ *as soon as* $1 \leq i \leq j \leq n$.

*A reduction sequence is said* **essentially standard** *if it is standard and that all the degrees involved in the sequence are finite.*

*We write* $t \longrightarrow_s^\star u$ *if there exists a standard reduction from* $t$ *to* $u$.

*Example 2.* let $t, u$ be closed $\Lambda\mu$-terms and consider the following reduction sequences (where $w_\alpha = w\{(v)x\alpha/(v)\alpha\}$, $w_\beta = w\{(v)y\beta/(v)\beta\}$, $w_{\alpha\beta} = (w_\alpha)_\beta$ and $w_{\alpha\beta\beta} = w_\alpha\{(v)yz\beta/(v)\beta\}$):

| | |
|---|---|
| $\sigma$ : $\mu\alpha.(\mu\beta.t)u\alpha$ | $\tau$ : $\mu\alpha.(\mu\beta.t)u\alpha$ |
| $\longrightarrow_{fst}$ $\lambda x.\mu\alpha.(\mu\beta.t_\alpha)u_\alpha x\alpha$ | $\longrightarrow_{fst}$ $\mu\alpha.(\lambda y.\mu\beta.t_\beta)u\alpha$ |
| $\longrightarrow_{fst}$ $\lambda x.\mu\alpha.(\lambda y.\mu\beta.t_{\alpha\beta})u_\alpha x\alpha$ | $\longrightarrow_{\beta_T}$ $\mu\alpha.(\mu\beta.t_\beta\{u/y\})\alpha$ |
| $\longrightarrow_{\beta_T}$ $\lambda x.\mu\alpha.(\mu\beta.t_{\alpha\beta}\{u_\alpha/y\})x\alpha$ | $\longrightarrow_{fst}$ $\lambda x.\mu\alpha.(\mu\beta.t_{\alpha\beta}\{u_\alpha/y\})x\alpha$ |
| $\longrightarrow_{fst}$ $\lambda x.\mu\alpha.(\lambda z.\mu\beta.t_{\alpha\beta\beta}\{u_\alpha/y\})x\alpha$ | $\longrightarrow_{fst}$ $\lambda x.\mu\alpha.(\lambda z.\mu\beta.t_{\alpha\beta\beta}\{u_\alpha/y\})x\alpha$ |
| $\longrightarrow_{\beta_T}$ $\lambda x.\mu\alpha.(\mu\beta.t_{\alpha\beta\beta}\{u_\alpha/y\}\{x/z\})\alpha$ | $\longrightarrow_{\beta_T}$ $\lambda x.\mu\alpha.(\mu\beta.t_{\alpha\beta\beta}\{u_\alpha/y\}\{x/z\})\alpha$ |

Then $\sigma$ is not standard since the first reduction step has degree $\omega$ while the second has degree 0 while $\tau$ is standard (the degrees are all equal to 0 in $\tau$).

The following lemma explains the terminology *superfluous* for $fst^s$-reductions:

**Lemma 3.** *In a standard reduction from a (closed) $\Lambda\mu$-term $t$ to a $\Lambda\mu$-term $u$ containing no $\mu$-abstraction (that is, a $\lambda$-term), there is no $fst^s$-reduction.*

We can now state the standardization theorem:

**Theorem 6 (Standardization).** *Standard reduction sequences always exist:*

$$\text{if } t \longrightarrow_{\beta fst}^\star u \text{ then } t \longrightarrow_s^\star u.$$

However, standard reductions are not unique. Indeed, there may exist several reduction sequences which are standard, equivalent (in some sense to be defined) though not equal. For instance reductions shown in example 1 are both standard and equivalent (and there is moreover reduction $\sigma_0 : \mu\alpha.(\mu\beta.x)\alpha \longrightarrow_{\beta_S} \mu\alpha.x \longrightarrow_{fst} \lambda y.\mu\alpha.x$ which is also standard though not essentially standard).

### 3.3 Strong Standardization

As a consequence, following Klop [18], we shall introduce a notion of strong standardization which shall ensure that there is a unique standard reductions sequence in a class of equivalent reductions. But first, we shall informally introduce the notion of equivalence on reductions that we shall consider and which is inspired from Lévy strong equivalence on reductions: using a standard indexing technique à la Lévy, adapted to $\Lambda\mu$-calculus, one can define residuals of a set of pre-redexes by a reduction sequence, $\mathcal{F}/\sigma$, and the residual of a reduction sequence by another reduction sequence, $\sigma/\tau$. Then, two reduction sequences $\sigma, \tau$ from $t$ to $u$ are equivalent is $\sigma/\tau = \tau/\sigma = \emptyset$.

Due to the more complex structure of $\Lambda\mu$-calculus, the description of strongly standard reductions is quite involved. We shall show a standardization procedure which transforms a reduction sequence into a strongly standard reduction sequence and use this to define strongly standard reductions.

**Definition 14 ($\mathbf{lmp}(\sigma), \mathbf{lmc}(\sigma)$).** *Let $\sigma = (\sigma_i)$ be a reduction from $t$ to $u$.*

- *If some p.r. of $t$ has a residual $r$ associated with $\sigma_n$ for some $n$, then $lmp(\sigma)$ is the leftmost such p.r. Otherwise, $\sigma$ is a $fst^s$-reduction sequence and $lmp(\sigma)$ is the leftmost fst-redex which undergoes a fst-reduction.*
- *$lmc(\sigma)$ is defined as follows:*
  - *if $lmp(\sigma)$ is a $(\mathcal{T})\mathcal{T}$, $(\mathcal{T})\mathcal{S}$ or $(\mathcal{S})\mathcal{T}$ p.r., then $lmc(\sigma)$ is the only reduction step that can be associated to this p.r.;*
  - *if $lmp(\sigma) = (\mu\alpha.t)\beta$, then $\beta_S$, fst on the binder of $\beta$ or fst on $\mu\alpha.t$ can be associated with this p.r. Thus we do a case analysis:*
    1. *if $lmp(\sigma)$ has no residual by $\sigma$, then it means that a $\beta_S$ is applied to some residual of $lmp(\sigma)$ and $lmc(\sigma)$ is this occurrence of $\beta_S$;*
    2. *otherwise, if there is in $\sigma$ a residual of $lmp(\sigma)$ of type $(\mathcal{T})\mathcal{T}$ or $(\mathcal{S})\mathcal{T}$, then $lmc(\sigma)$ is this occurrence of fst applied to the binder of $\beta$;*
    3. *otherwise, $lmc(\sigma)$ is the occurrence of fst applied to $\mu\alpha.t$;*
  - *if $lmp(\sigma)$ not a pre-redex, then it is a $\mu$-abstracted term and $lmc(\sigma)$ is the occurrence of fst applied to this $\mu$-abstraction.*
- *$p(\sigma) = \sigma/\{lmc(\sigma)\}$ is the residual of $\sigma$ after contracting $lmc(\sigma)$.*

**Definition 15 (Standardization process).** *Given a reduction sequence $\sigma : t = t_0 \xrightarrow{\sigma_1} t_1 \xrightarrow{\sigma_2} t_2 \xrightarrow{\sigma_3} t_3 \ldots \xrightarrow{\sigma_n} t_n$. One considers $\sigma_s$ defined as: $\sigma_s : t_0 \xrightarrow{lmc(\sigma)} t'_1 \xrightarrow{lmc(p(\sigma))} t'_2 \xrightarrow{lmc(p(p(\sigma)))} t'_3 \ldots$ ($\sigma_s$ stops when $p(p(\ldots p(\sigma))) = \emptyset$).*

*A* **strongly standard reduction** *$\sigma$ is a reduction which is invariant by the standardization process: $\sigma_s = \sigma$.*

**Theorem 7 (Strong Standardization).** *Let $\sigma$ be a finite reduction sequence.*

- *$\sigma_s$ is finite, equivalent to $\sigma$ and strongly standard;*
- *if $\tau$ is equivalent to $\sigma$, then $\sigma_s = \tau_s$.*

**Other Approaches to Standardization in $\lambda\mu$.** Py studied standardization for $\lambda\mu$-calculus in his PhD thesis. However, his approaches cannot be generalized and therefore our apporache to standardization differs quite radically from his. In particular, Py proves that it is possible to postpone applications of $\beta_S$ while this does not hold in the more general framework of $\Lambda\mu$-calculus: for instance, in $\mu\beta(\mu\alpha.\lambda x.(x)x)\beta\lambda x.(x)x$, the $(\Delta)\Delta$ cycling structure can appear only once a $\beta_S$ reduction has been applied.

As a consequence, our standard reductions have a very different structure from that of Py. Another important difference between the two approaches is that in our approach, the important notion is that of pre-redex rather than $\Lambda\mu$-redexes themselves. In our opinion, this makes more sense operationally in particular if one views $\Lambda\mu$ as a calculus computing on streams.

### 3.4 Solvability

We shall now characterize solvability in $\Lambda\mu$-calculus.

**Definition 16 (Stream applicative context).** *Stream applicative contexts are of the form* $[]t_1^1 \dots t_{n_1}^1 \alpha_1 \dots t_1^k \dots t_{n_k}^k \alpha_k$. *They are defined by:*

$$\mathcal{S} ::= [] \mid (\mathcal{S})t \mid (\mathcal{S})\alpha$$

**Definition 17.** $t \in \Sigma^c_{\Lambda\mu}$ *is solvable if there exists a stream applicative context* $\mathcal{S}$ *such that* $\mathcal{S}[t] \longrightarrow^\star_{\boldsymbol{\beta fst}} \lambda x.x$; $t \in \Sigma_{\Lambda\mu}$ *is solvable if its closure is solvable.*

The following theorem will be useful for characterizing solvability:

**Theorem 8.** *Let* $t \in \Sigma^c_{\Lambda\mu}$. *There exists a terminating head reduction path of* $t$ *iff* $t$ *has a head normal form.*

*Proof.* $\Rightarrow$ is trivial. For $\Leftarrow$, let us suppose $t$ has a hnf $h$ and consider a standard reduction sequence $\sigma = (t_i)_{0 \le i \le n}$ to $h$: $t \longrightarrow^\star_s h$. Since $\sigma$ is standard, $\sigma$ begins possibly with some head reduction steps and as soon as $t_i \longrightarrow t_{i+1}$ is an internal reduction all remaining reductions are internal, so that $t \longrightarrow^\star_h u \longrightarrow^\star_i h$. But since $h$ is in hnf, $u$ shall also be in hnf (otherwise $h$ would contain a head pre-redex) and $t \longrightarrow^\star_h u$ is the (finite) head reduction path for $t$. □

**Corollary 1.** *let* $t, u \in \Sigma_{\Lambda\mu}$, $\alpha \in \mathcal{V}_S$, $x \in \mathcal{V}_T$.

- $t$ *has a hnf iff* $\lambda x.t$ *has a hnf;*
- *if* $t$ *has no hnf then neither* $(t)u$, $(t)\alpha$, $t\{u/x\}$ *nor* $t\{(v)u\alpha/(v)\alpha\}$ *have a hnf.*

**Lemma 4.**
- $t \in \Sigma_{\Lambda\mu}$ *is solvable if, and only if, there exists a family of closed* $\Lambda\mu$*-terms* $(t_x)_{x \in FV_T(t)}$ *a family of vectors of closed* $\Lambda\mu$*-terms* $(\overrightarrow{t_\alpha})_{\alpha \in FV_S(t)}$ *and a closed stream applicative context* $\mathcal{S}$ *such that:*

$$\mathcal{S}\Big[t\{t_x/x\}\Big\{(v)\overrightarrow{t_\alpha}\alpha/(v)\alpha\Big\}\Big] \longrightarrow^\star \lambda x.x$$

- $t$ *is solvable iff* $\lambda x.t$ *is solvable iff* $\mu\alpha.t$ *is solvable.*

**Lemma 5.** *Let* $t, u \in \Sigma_{\Lambda\mu}$, $\alpha \in \mathcal{V}_S$, $x \in \mathcal{V}_T$. *If* $t$ *is unsolvable then* $(t)u$, $(t)\alpha$, $\lambda x.t$, $\mu\alpha.t$, $t\{u/x\}$ *and* $t\{(v)u\alpha/(v)\alpha\}$ *are also unsolvable.*

**Theorem 9 (Solvability).** $t \in \Sigma^c_{\Lambda\mu}$ *is solvable if, and only if, it has a hnf.*

*Proof.* $\boxed{\Rightarrow}$ Suppose that $t$ is solvable and $\mathcal{S}$ a stream applicative context such that $u = \mathcal{S}[t] \longrightarrow^\star \lambda x.x$. Thus $u$ has a hnf and $t$ as well by lemma 5.
$\boxed{\Leftarrow}$ Suppose $t \longrightarrow^\star \lambda\overrightarrow{x_0}.\mu\alpha_0.\dots.\mu\alpha_{n-1}.\lambda x_n^1 \dots x_n^k.(x_i^j)\overrightarrow{t_1}\beta_1 \dots \overrightarrow{t_m}\beta_m t^1 \dots t^l$.
Then if $\mathcal{S} = []\overrightarrow{u_0}\alpha_0 \dots \alpha_{n-1}u_n^1 \dots u_n^k$ with $u_i^j = \mu\gamma_1 \dots \gamma_m.\lambda z_1 \dots z_l.\lambda x.x$, one gets $\mathcal{S}[t] \longrightarrow^\star \lambda x.x$. □

## 4 Böhm trees for $\Lambda\mu$.

In this section, we introduce Böhm-like trees for $\Lambda\mu$-calculus. By doing so, we aim at making clearer the connections between $\Lambda\mu$-calculus and transfinitary $\lambda$-calculi. Moreover, those Böhm trees (and their corresponding Nakajima Trees, $\Lambda^n$-$\mathfrak{NT}$) are promising in at least two directions:

- getting more precise characterizations of separability for non-normalizing terms in the spirit of Barendregt-Dezani-Ronchi della Rocca results, semi-separability being characterized as ***compatibility*** of Nakajima trees;
- developing a Böhm model for $\Lambda\mu$-calculus based on these Böhm trees.

Moreover, we believe that these Böhm trees can be helpful in characterizing differences between languages by analyzing their characteristic ordinal. This might be a starting point for classifying the expressivity of those calculi by means of infinitary calculi (and to study the frontier between $\Lambda\mu$-calculus and $\lambda\mu$-calculus, that is between delimited and non-delimited control in CBN).

### 4.1 Stable Part of a $\Lambda\mu$-hnf.

**Definition 18.** *A $\Lambda\mu$-term $t$ will be said in* **Stream head normal form, shnf,** *when it is of the form $h = \lambda\overrightarrow{x_0}\mu\alpha_0 \dots \mu\alpha_n.(y)\overrightarrow{t_0}\beta_0 \dots \beta_m$.*

*Remark 2.* Every $\Lambda\mu$-hnf $t$ is $\eta_S$-equivalent to a shnf $u$. More precisely, if $t$ is a hnf, there exists a shnf $u$ such that $u \longrightarrow^{=}_{\eta_S} t$ (ie 0 or 1 reduction step).

An important property of hnf in $\lambda$-calculus is the following:
if $t = \lambda x_1 \dots \lambda x_n.(y)t_1 \dots t_m$, then $m, n$ and $y$ will remain identical on any $\beta$-reduction sequence of term $t$: if $t \longrightarrow^{\star}_{\beta} u$, then there exist $u_1 \dots u_m$ such that $t_i \longrightarrow^{\star}_{\beta} u_i$, for any $1 \leq i \leq m$ and $u = \lambda x_1 \dots \lambda x_n.(y)u_1 \dots u_m$.

Such a property cannot be directly transfered to $\Lambda\mu$-calculus. Indeed, because of *fst*-reduction, the size of vectors of variables $\overrightarrow{x_i}$ is not constant along $\boldsymbol{\beta}$**fst**-reduction sequences from a $\Lambda\mu$-hnf $h$ (actually the size of vectors $\overrightarrow{t_i}$ is not constant either). The following example makes this clear: $t = \lambda x.\mu\alpha.\lambda y.(x)\alpha y\alpha \longrightarrow^{\star}_{fst} \lambda x.\lambda x_1 \dots x_n.\mu\alpha.\lambda y.(x)x_1 \dots x_n\alpha y x_1 \dots x_n\alpha$. However, one can find a corresponding property which is stated in the following proposition:

**Proposition 1.** *Let $t \in \Sigma^c_{\Lambda\mu}$. Suppose that $t \longrightarrow^{\star}_{\boldsymbol{\beta}fst} h$ with $h = \lambda\overrightarrow{x_0}.\mu\alpha_0 \dots \mu\alpha_n.\lambda x^1_{n+1} \dots x^k_{n+1}.(y)\overrightarrow{t_0}\beta_0 \dots \beta_m t^1_{m+1} \dots t^l_{m+1}$, then $n, m, k, l$ and $y$ are characteristic of the head normal forms of $t$: if $t \longrightarrow^{\star}_{\boldsymbol{\beta}fst} h'$ with $h' = \lambda\overrightarrow{x'_0}\mu\alpha'_0 \dots \mu\alpha'_{n'}\lambda {x'}^1_{n'+1} \dots {x'}^{k'}_{n'+1}.(z)\overrightarrow{u_0}\beta'_0 \dots \beta'_{m'} u^1_{m'+1} \dots u^{l'}_{m'+1}$ then $m = m'$, $n = n'$, $k = k'$, $l = l'$ and $y = z$. Moreover, if $h \longrightarrow^{\star}_{\boldsymbol{\beta}fst} h'$, then $h'$ is $\lambda\overrightarrow{x_0}.\lambda\overrightarrow{x_{\alpha_0}}.\mu\alpha'_0 \dots \lambda\overrightarrow{x_n}.\lambda\overrightarrow{x_{\alpha_n}}.\mu\alpha'_n\lambda x^1_{n+1} \dots x^k_{n+1}.(y)\overrightarrow{t'_0}\overrightarrow{x_{\beta_0}}\beta'_0 \dots \overrightarrow{t'_m}\overrightarrow{x_{\beta_m}}\beta'_m {t'}^1_{m+1} \dots {t'}^l_{m+1}$. with $\overrightarrow{t_j} = t^1_j \dots t^{l_j}_j$, $\overrightarrow{t'_j} = {t'}^1_j \dots {t'}^{l_j}_j$, for $0 \leq j \leq m$ and $t^k_j\,\{(u)\overrightarrow{x_{\alpha_i}}\alpha'_i/(u)\alpha_i, i \leq n\} \longrightarrow^{\star}_{\boldsymbol{\beta}fst} {t'}^k_j$ for $0 \leq j \leq m, 1 \leq k \leq l_j$ or $j = m+1$ and $1 \leq k \leq l$.*

### 4.2 $\Lambda\mu$-Böhm Trees.

The previous property characterizes the stable information in a $\Lambda\mu$-hnf: up to possible *fst*-reductions, $\Lambda\mu$-head normal forms have essentially the same properties as $\lambda$-calculus hnf. In particular, to construct a Böhm-like tree structure for $\Lambda\mu$-calculus, one shall have an object that:

- records the relevant informations on $\Lambda\mu$-hnf: $(n, k)$ for the head abstractions, the head variable $y$, and the pair $(m, l)$ for the sons of the head;
- that is invariant by *fst*-reduction.

The following three observations lead us to the definition of the class of trees that will be called $\Lambda\mu$-Böhm trees:

- applying a *fst*-reduction to a hnf does not modify the five characteristic elements of the hnf mentioned in proposition 1;
- an arbitrary (finite) number of *fst*-reductions can be applied to any term containing a $\mu$-abstraction so that in the head abstraction of a head normal form, a $\mu$ can be preceded by an arbitrary number of $\lambda$ and in the argument branching, a stream application construction can be preceded by an arbitrary number of term applications;
- any ordinal $\lambda \in \omega^2$ is exactly characterized by a pair of natural numbers $(n, k)$ such that $\lambda = \omega.n + k$.

**Definition 19** ($\Lambda\mu$-$\mathfrak{BT}$). *Böhm trees for $\Lambda\mu$-calculus ($\mathfrak{B} \in \Lambda\mu$-$\mathfrak{BT}$) are (coinductively) defined as follows:*

$$\boxed{\mathfrak{B} ::= \Omega \mid \Lambda(x_i)_{i\in\mu\in\omega^2}.(y)(\mathfrak{B}_j)_{j\in\lambda\in\omega^2}}$$

One can now associate a $\Lambda\mu$-$\mathfrak{BT}$ to any (closed) $\Lambda\mu$-term thanks to the following definition:

**Definition 20.** *We define $\mathfrak{BT}^{\Lambda\mu} : \Sigma_{\Lambda\mu} \mapsto \Lambda\mu$-$\mathfrak{BT}$ as:*

- $\mathfrak{BT}^{\Lambda\mu}(t) := \Omega$ *if $t$ is unsolvable;*
- $\mathfrak{BT}^{\Lambda\mu}(t) := \Lambda(z_i)_{i\in\mu}.(y)(\mathfrak{BT}^{\Lambda\mu}(u_j))_{j\in\lambda}$
  *if* $t \longrightarrow \lambda\overrightarrow{x_1}\mu\alpha_1 \dots \mu\alpha_n\lambda\overrightarrow{x_{n+1}}.(y)\overrightarrow{t_1}\beta_1 \dots \beta_m\overrightarrow{t_{m+1}}$
  *with* $\overrightarrow{x_p} = x_p^1 \dots x_p^{k_p}$ *if* $1 \le p \le n+1$, *and* $\overrightarrow{t_p} = t_p^1 \dots t_p^{l_p}$ *if* $1 \le p \le m+1$
  *and with* $\mu = \omega.n + k_{n+1}$ *and* $\lambda = \omega.m + l_{m+1}$,
  - $z_{\omega.p+j} = \begin{cases} x_{p+1}^{j+1} & \text{if } 0 \le p \le n, 0 \le j < k_{p+1} \\ x_{\alpha_{p+1}}^{j-k_{p+1}} & \text{if } 0 \le p < n, k_{p+1} \le j < \omega \end{cases}$
  - $u_{\omega.p+j} = \begin{cases} t_{p+1}^{j+1} & \text{if } 0 \le p \le m, 0 \le j < l_{p+1} \\ x_{\beta_{p+1}}^{j-l_{p+1}} & \text{if } 0 \le p < m, l_{p+1} \le j < \omega \end{cases}$

**Remark 10** *The Böhm tree of a $\Lambda\mu$-term can also be obtained using direct approximants and completely (that is, infinitely) developing the $\mu$-abstractions thanks to fst.*

*Example 3.* Let $t = \mu\alpha.\lambda x.\mu\beta\lambda y.((x)y\ ((\Delta)\Delta)\beta)\ \beta$. $B = \Lambda(z_i)_{i\in\omega.2+1}.(z_\omega)(\mathfrak{B}_j)_{j\in\omega}$ with $\mathfrak{B}_0 = z_{\omega.2}$, $\mathfrak{B}_1 = \Omega$ and $\mathfrak{B}_{j+1} = z_{\omega.2+j}$ for $1 \le j < \omega$.

### 4.3 Nakajima Trees for $\Lambda\mu$-calculus.

We give a brief account of Nakajima Trees for $\Lambda\mu$-calculus. Nakajima Trees are infinite $\eta$-expanded Böhm trees. One can associate to any closed $\Lambda\mu$-term an infinite tree called a Nakajima tree:

**Definition 21** ($\Lambda\mu$-$\mathfrak{NT}$). *Nakajima trees for $\Lambda\mu$-calculus ($\mathfrak{N} \in \Lambda\mu$-$\mathfrak{NT}$) are (coinductively) defined as follows:* $\mathfrak{N} ::= \Omega \mid \Lambda(x_i)_{i\in\omega^2}.(y)(\mathfrak{B}_j)_{j\in\omega^2}$.

Two Nakajima trees $\mathcal{N}_1$ and $\mathcal{N}_2$ are said ***compatible*** if given a path $p$ in the trees (that is a finite word over the alphabet $\omega^2$), either the subtrees at $p$ have the same head variable or one of them at least is equal to $\Omega$. This can be used to obtain interesting result:

**Theorem 11.** *Given two closed $\Lambda\mu$-terms $u, v$, there exists an applicative a stream applicative context $\mathcal{S}$ such that $\mathcal{S}[u] \longrightarrow^\star_{\Lambda\mu} \lambda x, y.x$ and $\mathcal{S}[v] \longrightarrow^\star_{\Lambda\mu} \lambda x, y.y$ if, and only if, $\mathfrak{NT}^{\Lambda\mu}(u)$ and $\mathfrak{NT}^{\Lambda\mu}(v)$ are not compatible.*

### 4.4 A New Look at the Failure of Separation in $\lambda\mu$-calculus.

Parigot's $\lambda\mu$-calculus is a subset of $\Lambda\mu$-terms defined by the following grammar:

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

It is known that separation property fails in $\lambda\mu$-calculus. $\Lambda\mu$-Böhm trees provide a new look at this fact. Indeed, $\Lambda\mu$-$\mathfrak{BT}$ for $\lambda\mu$-terms have (up to finitely many $\eta_S$-expansions) the following particularly constrained shape:

$$\boxed{\mathfrak{B} ::= \Omega \mid x \mid \Lambda(x_i)_{i\in\omega}.(y)(\mathfrak{B}_j)_{j\in\omega}}$$

One can observe that there is no freedom on the arity in these Böhm trees: the index is always $\omega$ and the arities are forced to match. This is, in our opinion, the deep reason for the failure of separation in $\lambda\mu$-calculus. Below, we show a classification of several calculi (or logical formalisms) depending on whether they have the arity matching constraint, whether they are linear or not (single vs multiple occurrences of variables) and whether they have separation: ABT denotes Curien's Abstract Böhm Trees, $CPS_\infty$ denotes an infinitary languages studies by Streicher and Loew [21] and $\Lambda^n$ the languages of the Stream hierarchy [29] (see appendix). Interestingly one observes that Separation is obtained only when either arities don't match or when there is linearity.

| calculus | arity matching | non-linear | separation | (ref) |
|---|---|---|---|---|
| $\lambda$-calculus | no | yes | yes | [5] |
| $\lambda\mu$-calculus | yes | yes | no | [7] |
| $\Lambda\mu$-calculus | no | yes | yes | [30] |
| ABT | yes | yes | no | [22] |
| $\mathrm{CPS}_\infty$ | yes | yes | no | [21] |
| Ludics | yes | no | yes | [11] |
| $(\Lambda^n)_{n\in\omega}$ | no | yes | yes | [29] |

## 5 Conclusion, Perspectives and Future Works.

**Contributions of the Paper.** In the present paper, we introduced a notion of standard reductions for $\Lambda\mu$-calculus for which a standardization theorem (as well as a strong standardization theorem) holds. We then characterized solvability in $\Lambda\mu$-calculus thanks to the results on standardization. This allowed them to introduce a class of $\lambda\mu$-Böhm trees as transfinitary wide generalization of usual Böhm trees for $\lambda$-calculus. Those Böhm trees shed an interesting perspective on separation and non-separation in $\lambda\mu/\Lambda\mu$as well as in other calculi.

Those contributions are important in at least three directions:

- They make clear our understanding of head-normal forms in $\Lambda\mu$-calculus and allow for a more precise analysis of reductions in $\Lambda\mu$;
- They clarify the connections between $\Lambda\mu$ and infinitary $\lambda$-calculi;
- They open new perspectives for model-theoretic investigations of $\Lambda\mu$-calculus.

**Perspectives and Future Works.** The present works is connected with several other works and these connections shall be further investigated in the future:

- Lassen [20] provided a study of bisimulation relations in $\Lambda\mu$-calculus in which he proposed head-normal forms for $\Lambda\mu$-calculus as well as a characterization of solvability using this bisimilarity. Lassen's hnf are slightly different from the hnf we discussed here but, since solvability is characterized in terms of these hnf, they must be related to ours. We shall investigate this point;
- Loew [21] studied an infinitary version of $SPCF$, $SPCF_\infty$, as well as an infinitary target language, $CPS_\infty$. In particular, separation fails in $CPS_\infty$. Interestingly, infinite normal forms of $CPS_\infty$ are precisely $\lambda\mu$-$\mathfrak{BT}$.

Several other directions for future are exciting:

- We shall build a model of $\Lambda\mu$-calculus based on $\Lambda\mu$-$\mathfrak{BT}$;
- Not all $\Lambda\mu$-$\mathfrak{BT}$ are the Böhm tree of a $\Lambda\mu$-term (as phenomenon which is already observed with Nakajima trees for $\lambda$-calculus). We shall characterize those $\Lambda\mu$-Böhm trees which are the image of a $\Lambda\mu$-term;
- $\Lambda\mu$-$\mathfrak{BT}$ can be generalized to Böhm trees for the $\Lambda^n$-calculi of the Stream hierarchy [29]:

$$\boxed{\mathfrak{B} ::= \Omega \mid \Lambda(x_i)_{i\in\mu\in\omega^{n+1}}.(y)(\mathfrak{B}_j)_{j\in\lambda\in\omega^{n+1}}}$$

(Notice that the previous definition extends the definition for $\Lambda\mu$-Böhm trees ($n = 1$) as well as for $\lambda$-Böhm trees ($n = 0$).) We shall investigate those trees in the more general framework of the Stream Hierarchy (see appendix A);
- We geometry of the Böhm trees introduced in the present paper seems to say a lot about the relationships between various languages. In particular, we plan to investigate the frontier between delimited and non-delimited control thanks to these tools.

**Acknowledgments:** The author wishes to thank Simona Ronchi della Rocca as well as Kazushige Terui for helpful discussions on the material of this paper.

## References


1. Z. Ariola and H. Herbelin. Minimal classical logic and control operators. In *ICALP'03*, volume 2719 of *LNCS*. Springer, 2003.
2. K. Baba, S. Hirokawa, and K-E. Fujita. Parallel reduction in type free $\lambda\mu$-calculus. *ENTCS*, 42, 2001.
3. A. Berarducci. Infinite $\lambda$-calculus and non-sensible models. In Ursini and Aglianò, ed, no 180 in Lect. Notes in Pure and App. Math. M. Dekker Inc., 1996.
4. A. Berarducci and M. Dezani. Infinite $\lambda$-calculus and types. *TCS*, 212, 1999.
5. C. Böhm. Alcune proprietà delle forme $\beta\eta$-normali nel $\lambda K$-calcolo. *Publicazioni dell'Istituto per le Applicazioni del Calcolo*, 696, 1968.
6. O. Danvy and A. Filinski. Abstracting control. In *LISP & Fun. Prog.*, 1990.
7. R. David and W. Py. $\lambda\mu$-calculus and Böhm's theorem. *JSL*, 2001.
8. M. Dezani, P. Severi, and F-J. de Vries. Infinitary lambda calculus and discrimination of Berarducci trees. *TCS*, 2(298):275–302, 2003.
9. M. Felleisen, D P. Friedman, E E. Kohlbecker, and B F. Duba. A syntactic theory of sequential control. *TCS*, 52:205–237, 1987.
10. A. Filinski. Representing monads. In *POPL'94*, pages 446–457. ACM, 1994.
11. J-Y. Girard. Locus solum. *MSCS*, 11:301–506, 2001.
12. T. Griffin. A formulae-as-types notion of control. In *POPL'90*. IEEE, 1990.
13. H. Herbelin and S. Ghilezan. An approach to call-by-name delimited continuations. In *POPL'08*. ACM Sigplan, 2008.
14. W A. Howard. The formulae-as-type notion of construction. In *To Curry: Essays in Combinatory Logic, $\lambda$-Calculus, and Formalism*, p. 479–490. Acad. Press, 1980.
15. R. Kennaway, J W. Klop, M R. Sleep, and F-J. de Vries. Infinitary lambda calculi and böhm models. In *RTA*, pages 257–270, 1995.
16. Richard Kennaway, Jan Willem Klop, M. Ronan Sleep, and Fer-Jan de Vries. Infinitary lambda calculus. *TCS*, 175(1):93–125, 1997.
17. O. Kiselyov. Call-by-name linguistic side effects. In *ESSLLI 2008 Workshop on Symmetric calculi and Ludics for the semantic interpretation*, 2008.
18. J W. Klop. *Combinatory Reduction Systems*. Ph.D. thesis, Utrecht Univ., 1980.
19. J-L. Krivine. *Lambda-calculus, Types and Models*. Ellis Horwood, 1993.
20. S. Lassen. Hnf bisimulation for pairs and the $\lambda\mu$-calculus. In *LICS'06*, IEEE, 2006.
21. T. Loew. *Locally Boolean Domains and Universal Models for Infinitary Sequential Languages*. PhD thesis, Darmstadt University, 2006.
22. F. Maurel. *Un cadre quantitatif pour la Ludique*. PhD thesis, Univ. Paris 7, 2004.
23. M. Pagani and A. Saurin. SANE and $\Lambda\mu$-calculus. Tech. Rep. 6431, INRIA, 2008.
24. M. Parigot. Free deduction: An analysis of "computations" in classical logic. In *RCLP*, volume 592 of *LNAI*, pages 361–380, 1991. Springer-Verlag.
25. M. Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In *LPAR'92*, volume 624 of *LNCS*, London, UK, 1992. Springer-Verlag.
26. M. Parigot. Strong normalization for second order classical natural deduction. In M. Vardi, editor, *LICS'93*, pages 39–46. IEEE, June 1993.
27. M. Parigot. Proofs of strong normalisation for second order classical natural deduction. *Journal of Symbolic Logic*, 62(4):1461–1479, december 1997.
28. W. Py. *Confluence en $\lambda\mu$-calcul*. PhD thesis, Université de Savoie, 1998.
29. A. Saurin. A hierarchy for delimited continuations in call-by-name. submitted.
30. A. Saurin. Separation with streams in the $\Lambda\mu$-calculus. In *LICS'05*, p 356–365.
31. A. Saurin. On the relations between the syntactic theories of $\lambda\mu$-calculi. In *CSL 2008*, LNCS. Springer, September 2008.
32. A. Saurin. *Une étude logique du contrôle, appliquée à la programmation fonctionnelle et logique*. PhD thesis, École Polytechnique, September 2008.
33. A. Saurin. Typing streams in the $\Lambda\mu$-calculus. *ACM ToCL*, 2009. to appear.

# A Beyond $\Lambda\mu$-Böhm trees.

In this appendix, we briefly describe Böhm trees for the stream hierarchy which has been introduced in [29] as a generalization of $\Lambda\mu$-calculus which is a call-by-name version of the CPS hierarchy.

**Definition 22.** *Terms of the Stream hierarchy are defined using abstractions and variables with levels:*

| | | | |
|---|---|---|---|
| $\Sigma_{\Lambda^\omega}$ | $t,u ::= x^0 \mid \lambda^0 x.t \mid (t)u \mid \lambda^i x.t \mid (t)x^i$ | | *for any* $0 < i \leq n$ |
| $(\lambda^0 x.t)u$ | $\longrightarrow_{\beta^0}$ | $t\{u/x^0\}$ | |
| $(\lambda^i x.t)y^i$ | $\longrightarrow_{\beta^i}$ | $t\{y^i/x^i\}$ | *if* $0 < i \leq n$ |
| $\lambda^i x.(t)x^i$ | $\longrightarrow_{\eta^i}$ | $t$ | *if* $x^i \notin FV(t), 0 \leq i \leq n$ |
| $\lambda^i x.t$ | $\longrightarrow_{fst^{i/j}}$ | $\lambda^j x.\lambda^i x.t\{(v)x^j x^i/(v)x^i\}$ | *if* $x^j \notin FV(t)$ *and* $0 \leq j < i \leq n$ |

Böhm trees for the hierarchy generalize uniformly $\Lambda\mu$-Böhm trees:

**Definition 23.** *$\Lambda^n$-$\mathfrak{BT}$ are defined by the following inductive definition:*

$$\mathfrak{B} ::= \Omega \mid \Lambda(x_i)_{i\in\mu\in\omega^{n+1}}.(y)(\mathfrak{B}_j)_{j\in\lambda\in\omega^{n+1}}$$

Notice that the previous definition extends the definition for $\Lambda\mu$-Böhm trees ($n = 1$) as well as for $\lambda$-Böhm trees ($n = 0$).

# B More Details on Infinitary $\lambda$-calculi

Infinitary $\lambda$-calculus has been introduced independently by Berarducci [3] and by Kennaway et al. [16].

## B.1 Berarducci's infinite $\lambda$-calculus

Berarducci was interested in studying models of (finitary) $\lambda$-calculus which do not identify all the unsolvable terms (a *non-sensible model*). For this, he designs objects which are more precise than Böhm trees in the sense that they do not necessarily identify two unsolvable terms. This leads him to the definition of an infinitary version of $\lambda$-calculus built on infinite $\lambda$-trees and possibly infinite $\beta$-reduction sequences which converge in the following sense:

**Definition 24.** *Let $(t_i)_{i\in\omega}$ a sequence of (possibly infinite) terms such that for any $i \in \omega$, $t_i \longrightarrow_\beta t_{i+1}$. We say that $(t_i)_{i\in\omega}$ converges to a term $t$ if*

- *for any integer $k$, there exists an $n$ such that every $t_i$ for $i \geq n$ is identical to $t$ up to depth $k$;*
- *the depth of the reduction $t_i \longrightarrow_\beta t_{i+1}$ (ie. the depth of the $\beta$-redex) tends to infinity.*

Interestingly, Berarducci notices that there is no Böhm out technique for his infinite calculus.

### B.2 Kennaway et al's infinite $\lambda$-calculus

On the other hand, Kennaway *et al.* developed an infinitary version of $\lambda$-calculus as a generalization of their theory of infinitary rewriting of first-order infinitary terms. Their study is motivated by infinite structures which may occurs with lazy functional languages. Here, the definition of an infinite term depends on a definition of a depth on terms defined as follows (the definition of positions goes as usual in $\lambda$-calculus):

**Definition 25 (Depths $D^{abc}$).** *Let $a, b, c$ be elements in $\{0, 1\}$. Let $t$ be a term and $u$ be a position of $t$. Depth $D^{abc}(t, u)$ of th subterm of $t$ at position $u$ is defined as:*

- $D^{abc}(t, \langle\rangle) = 0$;
- $D^{abc}(\lambda x.t, 1 \cdot u) = a + D^{abc}(t, u)$;
- $D^{abc}((t_1)t_2, 1 \cdot u) = b + D^{abc}(t_1, u)$;
- $D^{abc}((t_1)t_2, 2 \cdot u) = c + D^{abc}(t_2, u)$.

*To a depth measure $D^{abc}$ is associated a distance $d^{abc}$ and the corresponding set of finite and infinite terms for this distance is noted $\Lambda^{abc}$.*

This approach identifies eight variants of infinite terms:

$$\Lambda^{000}, \Lambda^{001}, \Lambda^{010}, \Lambda^{011}, \Lambda^{100}, \Lambda^{101}, \Lambda^{110}, \Lambda^{111}$$

Berarducci calculus is $\Lambda^{111}$, the calculus associated with the lazy $\lambda$-calculus is $\Lambda^{101}$. The calculus associated with Parigot's $\lambda\mu$-calculus would correspond to $\Lambda^{11\star}$.

### B.3 $\Lambda\mu$-calculus and the Stream Hierarchy

The case of $\Lambda\mu$-calculus and the calculi of the stream hierarchy is slightly different from the previous calculi. While the calculi by Berarducci and Kennaway et al. allow for transfinite reduction sequences (for instance reduction sequences of length $\omega 2 + 1$), they only allow for infinite terms in which every subterm occurs at finite depth. On the contrary, $\Lambda\mu$-calculus and the stream hierarchy would lead to the consideration of terms of transfinite depths.

As Parigot observed, "the operator $\mu$ looks like a $\lambda$ having potentially infinite number of arguments" [25]. Phrased differently, the operator $\mu$ looks like an infinitary $\lambda$-abstraction while the construction $(t)\alpha$ looks like the application of $t$ to an infinite number of arguments:

- $\mu\alpha.t$ is considered as an ***abstraction over infinite streams of terms***

$$\mu(x_i^\alpha)_{i\in\omega}.t = \lambda x_1^\alpha \ldots x_n^\alpha \ldots .t$$

while

– $(t)\alpha$ is considered as the ***application of a term $t$ to an infinite stream of arguments***:

$$(t)[x_i^\alpha]_{i\in\omega} = (t)x_1^\alpha \dots x_n^\alpha \dots$$

The occurrence of terms of transfinite depth comes from the possibility, in $\Lambda\mu$-calculus, to consider terms of the form $\mu\alpha.\mu\beta.\lambda x.x.$ this term would correspond to the transfinite term $\lambda x_0, x_1 \dots x_\omega, x_{\omega+1} \dots x_{\omega 2}.x_{\omega 2}$.

Moving to the setting of the stream hierarchy, we can reach higher transfinite depth. For instance, $\lambda^2 x.\lambda^0 y.\lambda^1 z.\lambda^2 x'.\lambda^1 z'.\lambda^0 y'.(y^0)y'^0$ would correspond to $\lambda(x_i)_{i\in\omega^2 2+\omega+1}.(x_{\omega^2})x_{\omega^2 2+\omega}$.