Classical Call-by-need and duality

Zena M. Ariola[1] & Hugo Herbelin[2] & Alexis Saurin[2]

[1] University of Oregon, ariola@cs.uoregon.edu
[2] Laboratoire PPS, équipe πr^2, CNRS, INRIA & Université Paris Diderot
{herbelin,saurin}@pps.jussieu.fr

Abstract. We study call-by-need from the point of view of the duality between call-by-name and call-by-value. We develop sequent-calculus style versions of call-by-need both in the minimal and classical case. As a result, we obtain a natural extension of call-by-need with control operators. This leads us to introduce a call-by-need $\lambda\mu$-calculus. Finally, by using the dualities principles of $\overline{\lambda}\mu\tilde{\mu}$-calculus, we show the existence of a new call-by-need calculus, which is distinct from call-by-name, call-by-value and usual call-by-need theories.

Keywords: call-by-need, lazy evaluation, duality of computation, sequent calculus, $\lambda\mu$-calculus, classical logic, control, subtraction connective

Introduction

The theory of call-by-name λ-calculus [9,8] is easy to define. Given the syntax of λ-calculus $M ::= x \mid \lambda x.M \mid M\,M$, the reduction semantics is entirely determined by the β-reduction rule $(\lambda x.M)\,N \to_\beta M[x \leftarrow N]$ in the sense that:

- for any closed term M, either M is a value $\lambda x.N$ or M is a β-redex and for all $M \twoheadrightarrow V$, there is standard path $M \mapsto\!\!\!\!\twoheadrightarrow V'$ made only of β-redexes at the head, together with the property that $V' \twoheadrightarrow V$ using internal β-reductions;
- the observational closure of β induces a unique rule η that fully captures observational equality over finite normal terms (Böhm theorem);
- the extension with control, typically done à la Parigot [31], is relatively easy to get by adding just two operational rules and an observational rule (though the raw version of Böhm theorem fails [16,36]).

The theory of call-by-value λ-calculus, as initiated by Plotkin [32], has a similar property with respect to the β_v rule (the argument of β being restricted to a variable or a $\lambda x.M$ only), but the observational closure is noticeably more complex: it at least includes the rules unveiled by Moggi [28] as was shown by Sabry and Felleisen [35]. Extensions of standardization and Böhm theorem for call-by-value are more delicate than in call-by-name [25,34].

Comparatively, call-by-need λ-calculus, though at the core of programming languages such as Haskell [17], is at a much less advanced stage of development. The first approach to call-by-need as a calculus goes back to the 90's with the

works of Ariola *et al.* [3] and Maraist *et al.* [27] for whom the concern was the characterization of standard weak-head reduction. Our paper aims at studying call-by-need in terms that are familiar to the study of call-by-name and call-by-value. In particular, we will address the question of adding control to call-by-need and the question of what is the dual of call-by-need along the lines of the duality between call-by-name and call-by-value [21,37,11]. Call-by-need is close to call-by-value in the sense that only values are substituted, but call-by-need is also close to call-by-name in the sense that only those terms that are bound to needed variables are evaluated. In particular, with respect to evaluation of pure closed terms, the call-by-name and call-by-need calculi are not distinguishable. In order to tackle the problem of developing a classical version of call-by-need, we first study how to formulate (minimal) call-by-need in the sequent calculus setting [23] (while current call-by-need calculi are based on natural deduction [33]). An advantage of a sequent calculus presentation of a calculus is that its extension to the classical case does not require the introduction of new rules but simply the extension of existing ones [7].

Curien and Herbelin [11] designed a calculus that provides an appealing computational interpretation of proofs in sequent calculus, while providing at the same time a syntactic duality between terms, *i.e., producers*, and evaluation contexts, *i.e., consumers*, and between the call-by-name and call-by-value reduction strategies. By giving priority to the producer one obtains call-by-value, whereas by giving priority to the consumer one obtains call-by-name. In this paper, we present how call-by-need fits in the duality of computation. Intuitively, call-by-need corresponds to focusing on the consumer to the point where the producer is *needed*. The focus goes then to the producer till a value is reached. At that point, the focus returns to the consumer. We call this calculus *lazy call-by-value*, it is developed in Section 2 and 3. In addition to the properties of confluence and standardization, we show its correctness with respect to the call-by-name sequent calculus [11]. In Section 4, we develop the natural deduction presentation of call-by-need. The reduction theory is contained in the one of Maraist *et al.* [27] and extends the one of Ariola *et al.* [3]. Interestingly, the sequent calculus has suggested an alternative standard reduction which consists of applying some axioms (*i.e., lift* and *assoc*) eagerly instead of lazily. In Section 5, we show that the natural deduction and sequent calculus call-by-need are in reduction correspondence. In Section 6, we extend the minimal sequent calculus call-by-need with control, in both sequent calculus and natural deduction form. The calculi still enjoy confluence and standardization. The sequent calculus presentation of call-by-need naturally leads to a dual call-by-need, which corresponds to focusing on the producer and going to the consumer on a need basis. We call this calculus lazy call-by-name. In Section 7, we show how the dual call-by-need is obtained by dualizing the lazy call-by-value extended with the subtraction connective. We conclude and discuss our future work in Section 8. We start next with an overview of the duality of computation.

1 The duality of computation

Curien and Herbelin [11] provided classical sequent calculus with a term assignment, which is called the $\overline{\lambda}\mu\tilde{\mu}$ calculus. In $\overline{\lambda}\mu\tilde{\mu}$ there are two *dual* syntactic categories: terms which *produce* values and contexts which *consume* values. The interaction between a producer v and a consumer e is rendered by a command written as $\langle v \| e \rangle$, which is the computational counterpart of a sequent calculus cut. Contexts can be seen as evaluation contexts, that is, commands with a hole, written as $\Box$, standing for the term whose computation is to be done next: $\langle \Box \| e \rangle$. Thus, a command $\langle v \| e \rangle$ can be seen as filling the hole of the evaluation context e with v. Dually, terms can also be seen as commands with a *context* hole, standing for the context in which the term shall be computed. The duality of terms and contexts is also reflected at the variable level. One has two distinct sets of variables. The usual term variables $(x, y, \cdots)$ and the context variables $(\alpha, \beta, \cdots)$, which correspond to continuation variables. The set of terms, in addition to variables and lambda abstractions, contains a term of the form $\mu\alpha.c$, where c is a command, after Parigot's $\lambda\mu$-calculus [31]. The μ construct is similar to Felleisen's $\mathcal{C}$ control operator [19,20,18] and one can approximatively read $\mu\alpha.c$ as $\mathcal{C}(\lambda\alpha.c)$ (see [4] for a detailed analysis of the differences). Whereas the μ construct allows one to give a name to a context, so as to invoke it later, the dual construct, named $\tilde{\mu}$, allows one to name terms. One can read $\tilde{\mu}x.c$ as $\mathsf{let}\ x = \Box\ \mathsf{in}\ c$. Given a context e, $v \cdot e$ is also a context, which corresponds to an applicative context of the form $e[\Box v]$. The grammar of $\overline{\lambda}\mu\tilde{\mu}$ and its reduction theory are given below:

$$c ::= \langle v \| e \rangle \qquad v ::= x \mid \lambda x.v \mid \mu\alpha.c \qquad e ::= \alpha \mid \tilde{\mu}x.c \mid v \cdot e$$

$$\begin{array}{lll} (\beta) & \langle \lambda x.v \| v' \cdot e \rangle & \to \langle v' \| \tilde{\mu}x.\langle v \| e \rangle \rangle \\ (\mu) & \langle \mu\alpha.c \| e \rangle & \to c[\alpha \leftarrow e] \\ (\tilde{\mu}) & \langle v \| \tilde{\mu}x.c' \rangle & \to c'[x \leftarrow v] \end{array}$$

The reduction theory can be seen as consisting of structural reduction rules, μ and $\tilde{\mu}$, as well as logical reduction rules (here, only β, the rule corresponding to implication).

The calculus is not confluent due to a critical pair between μ and $\tilde{\mu}$:

$$\langle z \| \beta \rangle \leftarrow_{\mu} \langle \mu\alpha.\langle z \| \beta \rangle \| \tilde{\mu}x.\langle y \| \beta \rangle \rangle \rightarrow_{\tilde{\mu}} \langle y \| \beta \rangle$$

To regain confluence one can impose a strategy on how to resolve the critical pair $\mu/\tilde{\mu}$. By giving priority to the $\tilde{\mu}$ rule one captures *call-by-name*, whereas by giving priority to the μ rule one captures *call-by-value*. More generally, one can describe various ways to specialize the pair $\mu/\tilde{\mu}$ as reduction rules parametrized by sets $\mathcal{E}$ and $\mathcal{V}$, which denote sets of contexts and terms, respectively:

$$\begin{array}{lll} (\mu_{\mathcal{E}}) & \langle \mu\alpha.c \| e \rangle \to c[\alpha \leftarrow e] & \text{if } e \in \mathcal{E} \\ (\tilde{\mu}_{\mathcal{V}}) & \langle v \| \tilde{\mu}x.c' \rangle \to c'[x \leftarrow v] & \text{if } v \in \mathcal{V} \end{array}$$

This presentation with parametric rules is inspired by the work of Ronchi and Paolini on parametric λ-calculus [34]. A strategy corresponds to specifying which

contexts and terms can be duplicated or erased. For call-by-name, $\mathcal{V}$ is instantiated by V_n below but there are two possible choices E_n^{basic} and E_n for instantiating $\mathcal{E}$.

$$V_n ::= x \mid \lambda x.v \mid \mu\alpha.c \qquad E_n^{basic} ::= \alpha \mid v \cdot \tilde{\mu}x.c \qquad E_n ::= \alpha \mid v \cdot E_n$$

In both cases, this captures the fact that in call-by-name every term can be substituted for a term variable, while only specific contexts can be substituted for a context variable. In particular, the context $\tilde{\mu}x.c$ is not substitutable. The difference between the basic strategy and the second strategy is reminiscent of the difference between Plotkin's call-by-name continuation-passing-style semantics [32] and Lafont-Reus-Streicher's one [26]: the first one is not compatible with η but the second is. In the rest of the paper, we will consider the second strategy and the reduction rules corresponding to V_n and E_n are denoted as μ_n and $\tilde{\mu}_n$, respectively. We refer to E_n as an applicative context since it consists of a list of arguments. For call-by-value, the instantiations are V_v and E_v:

$$V_v ::= x \mid \lambda x.v \qquad E_v ::= \alpha \mid v \cdot E_v \mid \tilde{\mu}x.c$$

capturing the fact that only restricted terms (values) are substituted for a term variable, while every context can be substituted for a context variable. The resulting reduction rules are denoted as μ_v and $\tilde{\mu}_v$, respectively. Notice also that full non-deterministic $\overline{\lambda}\mu\tilde{\mu}$ corresponds to choosing μ_v together with $\tilde{\mu}_n$. As discussed next, call-by-need $\overline{\lambda}\mu\tilde{\mu}$-calculus will be defined with respect to another choice of parameters.

In addition to the instantiations of the structural rules $\mu_{\mathcal{E}}$ and $\tilde{\mu}_{\mathcal{V}}$, the calculi developed in the rest of the paper will contain rules for evaluating connectives. We will only consider implication, except in Section 7 where subtraction will also be added. We will also consider the following instances of the usual extensionality rules for μ and $\tilde{\mu}$ in $\overline{\lambda}\mu\tilde{\mu}$:

$$\begin{array}{ll} (\eta_{\mu}^{\mathcal{V}})\ \mu\alpha.\langle v \| \alpha\rangle \to v & v \in \mathcal{V} \text{ and } \alpha \text{ not free in } v \\ (\eta_{\tilde{\mu}}^{\mathcal{E}})\ \tilde{\mu}x.\langle x \| e\rangle \to e & e \in \mathcal{E} \text{ and } x \text{ not free in } e \end{array}$$

We denote the corresponding call-by-value and call-by-name instantiations as $\eta_{\mu}^{v}/\eta_{\tilde{\mu}}^{v}$ and $\eta_{\mu}^{n}/\eta_{\tilde{\mu}}^{n}$, respectively.

2 Call-by-need and duality

As we did for call-by-name and call-by-value, we have to specify the parametric sets used for call-by-need, that is, which terms and contexts can be substituted for term and context variables. Since call-by-need avoids duplication of work, it is natural to restrict the set $\mathcal{V}$ to V_v, thus allowing substitution of variables and lambda abstractions only. In order to specify which contexts are substitutable, it is important to notice that the goal of the structural rules is to bring to the top of a term the redex to be perfomed [2]. Thus, one should allow the reduction of

$\langle \mu\alpha.\underline{\langle I \| I \cdot \alpha\rangle} \| v \cdot \alpha\rangle$ (I stands for $\lambda x.x$) to $\langle I \| I \cdot v \cdot \alpha\rangle$ since the applicative redex (*i.e.,* the one underlined) is *needed* in order to continue the computation. This implies that E_n should be part of $\mathcal{E}$. That however is not enough. One would also want to reduce $\langle \mu\alpha.\langle I \| I \cdot \alpha\rangle \| \tilde{\mu}x.\langle x \| \alpha\rangle\rangle$ to $\langle I \| I \cdot \tilde{\mu}x.\langle x \| \alpha\rangle\rangle$. This however does not imply that $\tilde{\mu}x.c$ should be part of $\mathcal{E}$ since that would unveil an unwanted redex, such as in $\langle \mu\alpha.\langle I \| I \cdot \alpha\rangle \| \tilde{\mu}x.\langle z \| \alpha\rangle\rangle$. The only time we want to allow a change of focus from the consumer to the producer is when the producer is *needed*, which means that the variable bound to the producer occurs in the hole of a context; x is needed in $\langle x \| E_n\rangle$ but it is not needed in $\langle x \| \tilde{\mu}y.\langle z \| y \cdot \alpha\rangle\rangle$. This notion will still not capture a situation such as $\langle \mu\alpha.\langle I \| I \cdot \alpha\rangle \| \tilde{\mu}x.\langle v_1 \| \tilde{\mu}y.\langle x \| E_n\rangle\rangle\rangle$, since the needed variable is buried under the binding for y. This motivates the introduction of the notion of a call-by-need meta-context, which is simply a command hole surrounded by $\tilde{\mu}$-bindings:

$$C_l^{\tilde{\mu}} ::= \Box \mid \langle \mu\alpha.c \| \tilde{\mu}z.C_l^{\tilde{\mu}}\rangle$$

A variable x is needed in a command c, if c is of the form $C_l^{\tilde{\mu}}[\langle x \| E_n\rangle]$.

We have so far determined that $\mathcal{E}$ contains the call-by-name applicative contexts and contexts of the form $\tilde{\mu}x.C_l^{\tilde{\mu}}[\langle x \| E_n\rangle]$. This would allow the reduction of $\langle \mu\alpha.\langle I \| I \cdot \alpha\rangle \| \tilde{\mu}f.\langle f \| f \cdot \alpha\rangle\rangle$ to $\langle I \| I \cdot \tilde{\mu}f.\langle f \| f \cdot \alpha\rangle\rangle$. The problem is that the call-by-name applicative context considered so far does not contain a $\tilde{\mu}$. This is necessary to capture sharing. For example, in the above term $\langle I \| I \cdot \tilde{\mu}f.\langle f \| f \cdot \alpha\rangle\rangle$, the $\tilde{\mu}f$ captures the sharing of II. We need however to be careful about which $\tilde{\mu}$ we allow in the notion of applicative context. For example, we should disallow contexts such as $I \cdot \tilde{\mu}f.\langle z \| f \cdot \alpha\rangle$ since they might cause unwanted computation. Indeed, in the following reduction the application of I to I is computed while it is not needed to derive the result:

$$\langle I \| I \cdot \tilde{\mu}f.\langle z \| f \cdot \alpha\rangle\rangle \to_\beta \langle I \| \tilde{\mu}x.\langle x \| \tilde{\mu}f.\langle z \| f \cdot \alpha\rangle\rangle\rangle \to_{\tilde{\mu}_v} \langle I \| \tilde{\mu}f.\langle z \| f \cdot \alpha\rangle\rangle \to_{\tilde{\mu}_v} \langle z \| I \cdot \alpha\rangle.$$

This implies that a context $\tilde{\mu}x.c$ is allowed in an applicative context only if c demands x.

We are ready to instantiate the structural and extensional rules; $\mathcal{V}$ and $\mathcal{E}$ are instantiated as follows:

$$V_v ::= x \mid \lambda x.v \qquad E_l ::= F \mid \tilde{\mu}x.C_l^{\tilde{\mu}}[\langle x \| F\rangle] \quad \text{with} \quad F ::= \alpha \mid v \cdot E_l$$

resulting in reduction rules that we will denote as μ_l, $\tilde{\mu}_v$ and η_μ^v.

3 Minimal call-by-need in sequent calculus form ($\overline{\lambda}_{mlv}$)

A classical sequent calculus naturally provides a notion of control. However, one can restrict the calculus to be control-free by limiting the set of continuation variables to a single variable, conventionally written $\star$, which is linearly used. This corresponds to the restriction to minimal logic [1]. We introduce next the lazy call-by-value calculus, $\overline{\lambda}_{mlv}$.

Definition 1. *The syntax of $\overline{\lambda}_{mlv}$ is defined as follows:*

command	$c ::= \langle v \| e \rangle$
term	$v ::= x \mid \lambda x.v \mid \mu\star.c$
context	$e ::= E \mid \tilde{\mu}x.c$
yielding context	$E ::= F \mid \tilde{\mu}x.C[\langle x \| F \rangle]$
applicative context	$F ::= \star \mid v \cdot E$
meta-context	$C ::= \Box \mid \langle \mu\star.c \| \tilde{\mu}z.C \rangle$

The reduction of $\overline{\lambda}_{mlv}$, written as $\to_{mlv}$, denotes the compatible closure of β, μ_l, $\tilde{\mu}_v$ and η^v_μ; the relation $\twoheadrightarrow_{mlv}$ denotes the reflexive and transitive closure of $\to_{mlv}$ while $=_{mlv}$ denotes its reflexive, symmetric and transitive closure. The notion of weak head standard reduction is defined as:

$$\frac{c \to_\beta c'}{C[c] \mapsto_{mlv} C[c']} \qquad \frac{c \to_{\mu_l} c'}{C[c] \mapsto_{mlv} C[c']} \qquad \frac{c \to_{\tilde{\mu}_v} c'}{C[c] \mapsto_{mlv} C[c']}$$

The notation $\mapsto\!\!\!\!\twoheadrightarrow_{mlv}$ stands for the reflexive and transitive closure of $\mapsto_{mlv}$.
A weak head normal form (whnf) is a command c such that for no c', $c \mapsto_{mlv} c'$.

Notice how in the lazy call-by-value calculus, the standard redex does not necessarily occur at the top level. In $\langle v_1 \| \tilde{\mu}x_1.\langle v_2 \| \tilde{\mu}x_2.\langle \lambda x.v \| s \cdot \star \rangle \rangle \rangle$, the standard redex is buried under the bindings for x_1 and x_2, which is why the standard reduction refers to the meta-context. This however can be solved simply by going to a calculus with explicit substitutions, which would correspond to an abstract machine we are currently investigating. Some more discussions on this topic are available in section 8. Note that in a term of the form $\langle \lambda z.v \| \tilde{\mu}x.\langle x \| \tilde{\mu}y.\langle y \| \star \rangle \rangle \rangle$, the substitution for y is not the standard redex, and in

$$\langle \mu\star.\langle I \| I \cdot \star \rangle \| \tilde{\mu}x.\underline{\langle x \| \tilde{\mu}y.\langle y \| \star \rangle \rangle} \rangle \qquad \underline{\langle \mu\star.\langle V \| \tilde{\mu}y.\langle y \| \star \rangle \rangle \| \tilde{\mu}x.\langle x \| \star \rangle \rangle}$$

the standard redex is the underlined one. The η^v_μ rule is not needed for standard reduction. The η^v_μ rule turns a computation into a value, allowing for example the reduction: $\langle \mu\star.\langle V \| \star \rangle \| \tilde{\mu}x.\langle y \| x \cdot \star \rangle \rangle \to \langle V \| \tilde{\mu}x.\langle y \| x \cdot \star \rangle \rangle \to \langle y \| V \cdot \star \rangle$, which is not standard; in fact, the starting term is already in whnf.

Proposition 1 (Confluence). *$\to_{mlv}$ is confluent.*

Indeed, the only critical pair in $\overline{\lambda}_{mlv}$ is between η^v_μ and μ_l and it trivially converges since both rules produce the same resulting command.

Remark 1 In $\overline{\lambda}_{mlv}$ the duplicated redexes are all disjoint. This was not the situation in λ_{need} [27], where the *assoc* rule could have duplicated a *lift* redex. This does not happen in $\overline{\lambda}_{mlv}$ because the contexts are moved all at once, as described in the example below, which mimics the situation in λ_{need}.

$$\begin{array}{lll} \langle \mu\star\langle \mu\star.\langle z \| \tilde{\mu}y.\langle y \| \star \rangle \rangle \| \tilde{\mu}x.\langle x \| \star \rangle \rangle \| N \cdot \star \rangle & \to_{\mu_n} & \langle \mu\star.\langle z \| \tilde{\mu}y.\langle y \| \tilde{\mu}x.\langle x \| \star \rangle \rangle \rangle \| N \cdot \star \rangle \\ \quad \downarrow_{\mu_n} & & \quad \downarrow_{\mu_n} \\ \langle \mu\star.\langle z \| \tilde{\mu}y.\langle y \| \star \rangle \rangle \| \tilde{\mu}x.\langle x \| N \cdot \star \rangle \rangle & \to_{\mu_n} & \langle z \| \tilde{\mu}y.\langle y \| \tilde{\mu}x.\langle x \| N \cdot \star \rangle \rangle \rangle \end{array}$$

The needed constraint breaks the property that commands in weak head normal form are of the form $\langle x \| E \rangle$ or $\langle \lambda x.v \| \star \rangle$ (a property that holds for $\overline{\lambda}\mu\tilde{\mu}$ in call-by-name or call-by-value).

Definition 2. *Let $\boldsymbol{x}$ be a sequence of variables. $c_{\boldsymbol{x}}$ is defined by the grammar:*

$$c_{\boldsymbol{x}} ::= \langle \mu\star.c \| \tilde{\mu}y.c_{y\boldsymbol{x}} \rangle \mid \langle \lambda x.v \| \star \rangle \mid \langle z \| F \rangle \qquad z \notin \boldsymbol{x}.$$

Note that in $c_{\boldsymbol{x}}$, $\boldsymbol{x}$ records the variables which are $\tilde{\mu}$-bound to a computation on the path from the top of the term to the current position.

Proposition 2. *A command c is in weak head normal form iff it is in c_ϵ, where ϵ denotes the empty sequence of variables.*

Indeed, any command in c_ϵ is in whnf. Conversily, if c is in whnf, there is no c' such that $c \mapsto_{mlv} c'$ by definition, which means that c must be either of the form $C[\langle \lambda x.v \| \star \rangle]$ or $C[\langle z \| F \rangle]$ with z not bound by a $\tilde{\mu}$, otherwise said it must be in c_ϵ.

$\langle x \| \star \rangle$ is in whnf, however it is not of the form c_x since it demands variable x. Neither $\langle y \| \tilde{\mu}x.c \rangle$ nor $\langle \mu\star.c \| \tilde{\mu}x.\langle x \| \star \rangle \rangle$ are in whnf. A whnf is either of the form $C[\langle x \| F \rangle]$ or $C[\langle \lambda x.v \| \star \rangle]$. It is easy to see that standard redexes are mutually exclusive. For instance, a command c which had a standard β redex cannot have a μ_l or $\tilde{\mu}_v$ redex. Hence:

Proposition 3 (Unique Decomposition). *A command c is either a whnf or there exists a unique meta-context C and redex c' such that c is of the form $C[c']$.*

Using standard techniques (commutation of inner reductions with standard reduction), the following easily comes by induction:

Proposition 4 (Standardization). *Given a command c and a whnf c', if $c \twoheadrightarrow_{mlv} c'$ then there exists a whnf c'' such that $c \mapsto\!\!\!\twoheadrightarrow_{mlv} c''$ and $c'' \twoheadrightarrow_{mlv} c'$.*

3.1 Soundness and Completeness of $\overline{\lambda}_{mlv}$

The call-by-need calculus in natural deduction form is observationally equivalent to call-by-name. We show next that the same holds for call-by-need in sequent calculus form. To that end, we first review Curien and Herbelin call-by-name sequent calculus, called $\overline{\lambda}\mu\tilde{\mu}_T$ (after Danos *et al*'s LKT [12,13]). $\overline{\lambda}\mu\tilde{\mu}_T$ restricts the syntax of legal contexts capturing the intuition that according to the call-by-name continuation passing style, the continuation follows a specific pattern. The syntax of $\overline{\lambda}\mu\tilde{\mu}_T$ becomes:

$$c ::= \langle v \| e \rangle \qquad v ::= V_n \qquad e ::= \tilde{\mu}x.c \mid E_n$$

Notice that whereas $v \cdot \tilde{\mu}x.c$ is a legal context in $\overline{\lambda}\mu\tilde{\mu}$, it is not legal in $\overline{\lambda}\mu\tilde{\mu}_T$. The reduction theory of $\overline{\lambda}\mu\tilde{\mu}_T$ consists of β, μ_n and $\tilde{\mu}_n$.

The $\overline{\lambda}_{mlv}$ calculus is sound and complete with respect to the minimal restriction of $\overline{\lambda}\mu\tilde{\mu}_T$. We first need to translate $\overline{\lambda}_{mlv}$ terms to $\overline{\lambda}\mu\tilde{\mu}_T$ terms by giving a

name to the $\tilde{\mu}$-term contained in a linear context. The translation, written as $(.)^\circ$, is defined as follows (the interesting cases of the translation are the last two cases), with $n \geq 0$:

$$\begin{aligned}
x^\circ &= x \\
(\lambda x.v)^\circ &= \lambda x.v^\circ \\
(\mu\star.c)^\circ &= \mu\star.c^\circ \\
(\langle v \| w_1 \cdot \ldots w_n \cdot \star\rangle)^\circ &= \langle v^\circ \| w_1^\circ \cdot \ldots w_n^\circ \cdot \star\rangle \\
(\langle v \| \tilde{\mu} x.c\rangle)^\circ &= \langle v^\circ \| \tilde{\mu} x.c^\circ\rangle \\
(\langle v \| w_0 \cdot \ldots w_n \cdot \tilde{\mu} x.c\rangle)^\circ &= \langle \mu\star.\langle v^\circ \| w_0^\circ \cdot \ldots w_n^\circ \cdot \star\rangle \| \tilde{\mu} x.c^\circ\rangle
\end{aligned}$$

we then have the following properties:

Lemma 1. *If c is a command in $\overline{\lambda}_{mlv}$, then c° is a command in $\overline{\lambda}\mu\tilde{\mu}_T$.*

The previous lemma holds since the syntactical constraint on $\overline{\lambda}\mu\tilde{\mu}_T$ commands is that a context is either of the form $\tilde{\mu}x.c$ or it is a stack a terms pushed on top of $\star$: the translation precisely achieves this goal.

Proposition 5. *(i) Given a $\overline{\lambda}_{mlv}$ term v, $v =_{mlv} v^\circ$.*
(ii) Given terms v and w in $\overline{\lambda}\mu\tilde{\mu}_T$, v and w are also in $\overline{\lambda}_{mlv}$ and we have:
(a) $v =_{mlv} w$ then $v =_{\overline{\lambda}\mu\tilde{\mu}_T} w$;
(b) $v =_{\overline{\lambda}\mu\tilde{\mu}_T} (\lambda x.w\|\star)$ then $v =_{mlv} C[(\lambda x.w'\|\star)]$ for some C and w'.

Indeed, $\overline{\lambda}_{mlv}$ theory restricted to the call-by-name syntax of $\overline{\lambda}\mu\tilde{\mu}_T$ is included in $\overline{\lambda}\mu\tilde{\mu}_T$ theory.

Intermezzo 2 Soundness can also be shown with respect to the $\overline{\lambda}\mu\tilde{\mu}$ calculus without the need of doing a translation, since the $\overline{\lambda}\mu\tilde{\mu}$ calculus does not impose any restrictions on the context. This however requires extending the $\tilde{\mu}$ rule to $\langle v \| v_1 \cdots v_n.\tilde{\mu}x.c\rangle \to c[x = \mu\star.\langle v \| v_1 \cdots v_n.\star\rangle]$. The rule is sound for call-by-name extended with the eta rule, called $\eta^R_{\to}$ in [24], given as $y = \lambda x.\alpha.\langle y \| x.\alpha\rangle$. We have:

$$\begin{aligned}
\langle v \| w \cdot \tilde{\mu}x.c\rangle &=_{\tilde{\mu}} \langle v \| \tilde{\mu}y.\langle y \| w \cdot \tilde{\mu}x.c\rangle\rangle \\
&=_{\eta^R_{\to}} \langle v \| \tilde{\mu}y.\langle \lambda z.\mu\star.\langle y \| z \cdot \star\rangle \| w \cdot \tilde{\mu}x.c\rangle\rangle \\
&=_{\to} \langle v \| \tilde{\mu}y.\langle w \| \tilde{\mu}z.\langle \mu\star.\langle y \| z \cdot \star\rangle \| \tilde{\mu}x.c\rangle\rangle\rangle \\
&= \tilde{\mu} \langle v \| \tilde{\mu}y.\langle w \| \tilde{\mu}z.c[x = \mu\star.\langle y \| z \cdot \star\rangle]\rangle\rangle \\
&= \tilde{\mu} \langle v \| \tilde{\mu}y.c[x = \mu\star.\langle y \| w \cdot \star\rangle]\rangle \\
&= \tilde{\mu} c[x = \mu\star.\langle v \| w \cdot \star\rangle]
\end{aligned}$$

4 Minimal call-by-need in Natural Deduction ($\boldsymbol{\lambda_{need}}$)

We now present a natural deduction counterpart to $\overline{\lambda}_{mlv}$.

Definition 3. *The syntax of λ_{need} is defined as follows:*

term	M	$::= V \mid M_{nv}$
value	V	$::= x \mid \lambda x.M$
computation	M_{nv}	$::= MM \mid \mathsf{let}\ x = M\ \mathsf{in}\ N$
applicative context	C_{ap}	$::= \Box \mid C_{ap}M$
needed context	C	$::= C_{ap} \mid \mathsf{let}\ x = M_{nv}\ \mathsf{in}\ C \mid \mathsf{let}\ x = C_{ap}M\ \mathsf{in}\ C[x]$

Reduction in λ_{need}, written as $\rightarrow_{need}$, is the compatible closure of the following rules:

(β)	$(\lambda x.N)M$	$\rightarrow \mathsf{let}\ x = M\ \mathsf{in}\ N$
$(lift)$	$(\mathsf{let}\ x = M\ \mathsf{in}\ P)N$	$\rightarrow \mathsf{let}\ x = M\ \mathsf{in}\ PN$
$(deref_v)$	$\mathsf{let}\ x = V\ \mathsf{in}\ M$	$\rightarrow M[x \leftarrow V]$
$(assoc)$	$\mathsf{let}\ z = (\mathsf{let}\ x = M\ \mathsf{in}\ N)\ \mathsf{in}\ C[z]$	$\rightarrow \mathsf{let}\ x = M\ \mathsf{in}\ \mathsf{let}\ z = N\ \mathsf{in}\ C[z]$

The relation $\twoheadrightarrow_{need}$ denotes the reflexive and transitive closure of $\rightarrow_{need}$.
The notion of weak head standard reduction is defined as:

$$\frac{M \rightarrow_{\beta,lift} N}{C_{\beta l}[M] \mapsto_{need} C_{\beta l}[N]} \qquad \frac{M \rightarrow_{deref_v,assoc} N}{C_{da}[M] \mapsto_{need} C_{da}[N]}$$

where

$$C_{\beta l} ::= C_{ap} \mid \mathsf{let}\ x = M_{nv}\ \mathsf{in}\ C_{\beta l} \mid \mathsf{let}\ x = C_{ap}\ \mathsf{in}\ C[x]$$
$$C_{da} ::= \Box \mid \mathsf{let}\ x = M_{nv}\ \mathsf{in}\ C_{da}$$

The notation $\mapsto\!\!\!\!\twoheadrightarrow_{need}$ stands for the reflexive and transitive closure of $\mapsto_{need}$.
A weak head normal form (whnf) is a term M such that for no N, $M \mapsto_{need} N$.

Unlike the calculi defined by Maraist *et al.* [27] and Ariola *et al.* [3], the $deref_v$ rule follows the call-by-value discipline since it substitutes a value for each occurrence of the bound variable, even if the variable is not needed. The rule is derivable in the calculus of Maraist *et al.* using garbage collection. The *assoc* rule is more constrained than in the calculus of Maraist *et al.* since it performs the flattening of the bindings on a demand basis. The *assoc* requires the variable to appear in the hole of a context C, whose definition does not allow a hole to be bound to a let variable. For example, $\mathsf{let}\ x = \Box\ \mathsf{in}\ x$ and $\mathsf{let}\ x = \Box\ \mathsf{in}\ \mathsf{let}\ y = x\ \mathsf{in}\ y$ are not C contexts. This restriction is necessary to make sure that in a term of the form

$$\mathsf{let}\ x = (\mathsf{let}\ z = N\ \mathsf{in}\ P)\ \mathsf{in}\ \mathsf{let}\ y = x\ \mathsf{in}\ y$$

the standard redex is the substitution for y and not the *assoc* redex. The *assoc* rule is more general than in [3], since it does not require the binding for z to be an answer (*i.e.,* an abstraction surrounded by bindings). The *lift* rule is the same as in [27], it is more general than the corresponding rule in [3] since the left-hand side of the application is not restricted to be an answer. The calculi in [27] and [3] share the same standard reduction. For example, in the terms:

$$(\mathsf{let}\ y = M\ \mathsf{in}\ (\lambda x.x)y)P \qquad \mathsf{let}\ y = (\mathsf{let}\ z = N\ \mathsf{in}\ (\lambda x.x)y)\ \mathsf{in}\ y$$

$(\lambda x.x)y$ is the standard redex. Our standard reduction differs. The above terms correspond to a *lift* and *assoc* redex, respectively. Moreover, our standard reduction is also defined for open terms. Thus, the following terms:

$$(\mathsf{let}\ y = xz\ \mathsf{in}\ y)P \qquad \mathsf{let}\ y = (\mathsf{let}\ z = xP\ \mathsf{in}\ z)\ \mathsf{in}\ y$$

instead of being of the form $C[x]$, reduce further. The standard reduction requires different closure operations to avoid the interference between reductions. In

$$\mathsf{let}\ z = (\mathsf{let}\ x = V\ \mathsf{in}\ N)\ \mathsf{in}\ z \qquad\qquad \mathsf{let}\ y = (\mathsf{let}\ z = (\mathsf{let}\ x = M\ \mathsf{in}\ N)\ \mathsf{in}\ P)\ \mathsf{in}\ y$$

the standard redex is the (outermost) *assoc*, and in $\mathsf{let}\ x = II\ \mathsf{in}\ \mathsf{let}\ y = x\ \mathsf{in}\ y$, the $deref_v$ is the standard redex.

Proposition 6 (Confluence). *$\to_{need}$ is confluent.*

This is because all critical pairs converge.

Proposition 7 (Unique Decomposition). *A term M is either a whnf or there exists a unique $C_{\beta l}$ such that M is of the form $C_{\beta l}[P]$, where P is a β or lift redex, or there exists a unique C_{da} such that M is of the form $C_{da}[P]$, where P is a $deref_v$ or assoc redex.*

The previous proposition essentially relies on the facts that $C[x]$ is a whnf and that $C_{da} \subset C \subset C_{\beta l}$.

Proposition 8 (Standardization). *Given a term M and whnf N, if $M \twoheadrightarrow_{need} N$ then there exists a whnf N' such that $M \mapsto\!\!\!\twoheadrightarrow_{need} N'$ and $N' \twoheadrightarrow_{need} N$.*

Definition 4. *Let $\boldsymbol{x}$ be a sequence of variables. $M_{\boldsymbol{x}}$ is defined as:*

$$\begin{array}{rl} M_{\boldsymbol{x}} ::= \lambda x.N & \mid \mathsf{let}\ y = N_{nv}\ \mathsf{in}\ M_{y\boldsymbol{x}} \\ \mid zN_1 \cdots N_n & \mid \mathsf{let}\ y = zNN_1 \cdots N_n\ \mathsf{in}\ C[y] \qquad z \notin \boldsymbol{x} \end{array}$$

Proposition 9. *A term M is in whnf iff it is in M_ϵ (with ϵ the empty sequence).*

4.1 Soundness and completeness of $\boldsymbol{\lambda_{need}}$

Our calculus is sound and complete for evaluation to an answer (*i.e.,* an abstraction or a let expression whose body is an answer) with respect to the standard reduction of the call-by-need calculi defined in [27] and [3], denoted by $\mapsto\!\!\!\twoheadrightarrow^{af}_{mow}$.

Proposition 10. *Let M be a term and A be an answer.*

- *If $M \mapsto\!\!\!\twoheadrightarrow_{need} A$ then there exists an answer A' such that $M \mapsto\!\!\!\twoheadrightarrow^{af}_{mow} A'$;*
- *If $M \mapsto\!\!\!\twoheadrightarrow^{af}_{mow} A$ then there exists an answer A' such that $M \mapsto\!\!\!\twoheadrightarrow_{need} A'$.*

Indeed, the discussion at the beginning of the section evidences that $\to_{need}$ is contained in $\to_{mow}$ and contains $\to_{af}$ and the result follows from the fact that standard reductions of $\to_{mow}$ and $\to_{af}$ coincide.

5 Correspondence between $\overline{\lambda}_{mlv}$ and λ_{need}

The calculi $\overline{\lambda}_{mlv}$ and λ_{need} are in reduction correspondence for the following translations from λ_{need} to $\overline{\lambda}_{mlv}$ and vice-versa:

Definition 5. *Given a term M in λ_{need}, a term v, a context e and a command c in $\overline{\lambda}_{mlv}$, translations $M^{\triangleright}$, $M^{\triangleright}_e$, $v^{\triangleleft}$, $e^{\triangleleft}$ and $c^{\triangleleft}$ are defined as follows:*

$$
\begin{array}{lll}
x^{\triangleright} & = x & \langle v \| e \rangle^{\triangleleft} = e^{\triangleleft}[v^{\triangleleft}] \\
(\lambda x.M)^{\triangleright} & = \lambda x.M^{\triangleright} & x^{\triangleleft} = x \\
(MN)^{\triangleright} & = \mu\star.(MN)^{\triangleright}_{\star} & (\lambda x.v)^{\triangleleft} = \lambda x.v^{\triangleleft} \\
(\mathsf{let}\ x = M\ \mathsf{in}\ N)^{\triangleright} & = \mu\star.(\mathsf{let}\ x = M\ \mathsf{in}\ N)^{\triangleright}_{\star} & (\mu\star.c)^{\triangleleft} = c^{\triangleleft} \\
(MN)^{\triangleright}_e & = M^{\triangleright}_{N^{\triangleright}.e} & \star^{\triangleleft} = \Box \\
(\mathsf{let}\ x = M\ \mathsf{in}\ N)^{\triangleright}_e & = \begin{cases} M^{\triangleright}_{\tilde{\mu}x.N^{\triangleright}_e} & N \equiv C[x] \\ \langle M^{\triangleright} \| \tilde{\mu}x.N^{\triangleright}_e \rangle & \textit{otherwise} \end{cases} & (v \cdot E)^{\triangleleft} = E^{\triangleleft}[\Box v^{\triangleleft}] \\
V^{\triangleright}_e & = \langle V^{\triangleright} \| e \rangle & (\tilde{\mu}x.c)^{\triangleleft} = \mathsf{let}\ x = \Box\ \mathsf{in}\ c^{\triangleleft}
\end{array}
$$

We first illustrate the correspondence on an example.

Example 3 Consider the following λ_{need} reduction, where I stands for $\lambda y.y$ and M for $(\lambda f.fI(fI))((\lambda z.\lambda w.zw)(II))$:

$$
\begin{array}{lll}
M \to_{\beta} \mathsf{let} & \to_{\beta} \mathsf{let} & \to_{assoc} \mathsf{let} \\
\quad f = (\lambda z.\lambda w.zw)(II) & \quad f = \mathsf{let} & \quad z = II \\
\quad \mathsf{in}\ fI(fI) & \quad\quad z = II & \quad \mathsf{in}\ \mathsf{let} \\
 & \quad\quad \mathsf{in}\ \lambda w.zw & \quad\quad f = \lambda w.zw \\
 & \quad \mathsf{in}\ fI(fI) & \quad\quad \mathsf{in}\ fI(fI)
\end{array}
$$

We have $M^{\triangleright}_{\star} = \langle \lambda f.\mu\star.\langle f \| I \cdot (fI)^{\triangleright} \cdot \star \rangle \| \mu\star.\langle \lambda z.\lambda w.(zw)^{\triangleright} \| (II)^{\triangleright} \cdot \star \rangle \cdot \star \rangle$. The first β step is simulated by the following $\overline{\lambda}_{mlv}$ reduction, where we underline the redex to be contracted unless it occurs at the top:

$$
\begin{array}{l}
\langle \lambda f.\mu\star.\langle f \| I \cdot (fI)^{\triangleright} \cdot \star \rangle \| \mu\star.\langle \lambda z.\lambda w.(zw)^{\triangleright} \| (II)^{\triangleright} \cdot \star \rangle \cdot \star \rangle \to_{\beta} \\
\langle \mu\star.\langle \lambda z.\lambda w.(zw)^{\triangleright} \| (II)^{\triangleright} \cdot \star \rangle \| \tilde{\mu}f.\underline{\langle \mu\star.\langle f \| I \cdot (fI)^{\triangleright} \cdot \star \rangle \| \star \rangle} \rangle \to_{\mu_l} \\
\langle \mu\star.\langle \lambda z.\lambda w.(zw)^{\triangleright} \| (II)^{\triangleright} \cdot \star \rangle \| \tilde{\mu}f.\underline{\langle f \| I \cdot (fI)^{\triangleright} \cdot \star \rangle} \rangle \to_{\mu_l} \\
\langle \lambda z.\lambda w.(zw)^{\triangleright} \| (II)^{\triangleright} \cdot \tilde{\mu}f.\langle f \| I \cdot (fI)^{\triangleright} \cdot \star \rangle \rangle
\end{array}
$$

The second μ_l step corresponds to moving the redex in the context $\mathsf{let}\ f = \Box\ \mathsf{in}\ C[f]$ at the top. The simulation of the second β step leads to:

$$\langle (II)^{\triangleright} \| \tilde{\mu}z.\langle \lambda w.(zw)^{\triangleright} \| \tilde{\mu}f.\langle f \| I \cdot (fI)^{\triangleright} \cdot \star \rangle \rangle \rangle$$

The *assoc* corresponds to an identity in $\overline{\lambda}_{mlv}$.

Notice that the restriction on the *assoc* rule is embedded in the sequent calculus. The simulation of a non restricted *assoc* would require a generalization of the μ_l rule. For example, the simulation of the reduction:

$$\mathsf{let}\ x = (\mathsf{let}\ y = II\ \mathsf{in}\ y)\ \mathsf{in}\ 0 \to \mathsf{let}\ y = II\ \mathsf{in}\ \mathsf{let}\ x = y\ \mathsf{in}\ 0$$

would require equating the following terms:

$$\langle \mu\star.\langle I \| I \cdot \tilde{\mu}y.\langle y \| \star\rangle\rangle \| \tilde{\mu}x.\langle 0 \| \star\rangle\rangle = \langle \mu\star.\langle I \| I \cdot \star\rangle \| \tilde{\mu}y.\langle y \| \tilde{\mu}x.\langle 0 \| \star\rangle\rangle\rangle$$

However, those should not be equated to $\langle I \| I \cdot \tilde{\mu}y.\langle y \| \tilde{\mu}x.\langle 0 \| \star\rangle\rangle\rangle$. That would correspond to relaxing the restriction of E_l in the μ_l rule, and has the problem of bringing the redex II to the top and thus becoming the standard redex.

Proposition 11 (Simulation). *Call-by-need reduction in natural deduction and sequent calculus form are in reduction correspondence:*
(i) $M \twoheadrightarrow_{need} M^{\triangleright\triangleleft}$
(ii) $c \twoheadrightarrow_{mlv} c^{\triangleleft\triangleright}$
(iii) If $M \to_{need} N$ *then* $M^{\triangleright} \twoheadrightarrow_{mlv} N^{\triangleright}$
(iv) If $c \to_{mlv} c'$ *then* $c^{\triangleleft} \twoheadrightarrow_{need} c'^{\triangleleft}$

Remark 4 Note that the translation $(\_)^{\triangleright}_{e}$ of a let expression depends on the bound variable being needed or not. The choice of this *optimized* translation was required to preserve reduction. Indeed, otherwise, to simulate the *assoc* reduction one would need an expansion in addition to a reduction.

6 Classical call-by-need in sequent calculus ($\overline{\lambda}_{lv}$) and natural deduction form ($\lambda\mu_{need}$)

Defining sequent calculus classical call-by-need, called $\overline{\lambda}_{lv}$, requires extending the applicative context and the μ construct to include a generic continuation variable[3]. The syntax of $\overline{\lambda}_{lv}$ becomes:

$$\begin{array}{ll} c ::= \langle v \| e\rangle & E ::= F \mid \tilde{\mu}x.C[\langle x \| F\rangle] \\ v ::= x \mid \lambda x.v \mid \mu\alpha.c & F ::= \alpha \mid v \cdot E \\ e ::= E \mid \tilde{\mu}x.c & C ::= \Box \mid \langle \mu\alpha.c \| \tilde{\mu}z.C\rangle \end{array}$$

Reduction, weak head standard reduction (written as $\to_{lv}$ and $\mapsto_{lv}$, respectively) and weak head normal form (whnf) are defined as in the minimal case by replacing $\star$ with any context variable α. For example, a term of the form $\langle \mu\alpha.\langle x \| \beta\rangle \| \tilde{\mu}x.\langle y \| y \cdot \delta\rangle\rangle$ is in weak head normal form.

Unique decomposition, confluence and standardization extend to the classical case. Once control is added to the calculus, call-by-need and call-by-name are observationally distinguishable, as witnessed by the example given in the next section. It is important to notice that the bindings are not part of the captured context. For example, in the following command, the redex II written as $\mu\alpha.\langle \lambda x.x \| (\lambda x.x) \cdot \alpha\rangle$ will be executed only once. Whereas, if the bindings were part of the captured context then that computation would occur twice.

$$\langle II \| \tilde{\mu}z.\langle \mu\alpha.\langle \lambda x.\mu\beta.\langle z \| (\mu\delta.\langle \lambda x.x \| \alpha\rangle) \cdot \beta\rangle \| \alpha\rangle \| \tilde{\mu}f.\langle f \| z \cdot \gamma\rangle\rangle\rangle\rangle$$

[3] To reduce closed commands one can introduce a constant named **tp** as in [5], or one can encode the top-level using subtraction (see Section 7).

Unlike the sequent calculus setting, to extend minimal natural deduction to the classical case, we need to introduce two new constructs: the capture of a continuation and the invocation of it, written as $\mu\alpha.J$ and $[\alpha]M$, where J stands for a jump (*i.e.,* an invocation of a continuation). The reduction semantics makes use of the notion of *structural substitution*, which was first introduced in [31] and is written as $J[\alpha \leftarrow [\alpha]F]$ indicating that each occurrence of $[\alpha]M$ in J is replaced by $[\alpha]F[M]$, where F is the context captured by a continuation which is either $\Box\, M$ or $\mathsf{let}\ x = \Box\ \mathsf{in}\ C[x]$. The benefits of structural substitution are discussed in [4]. In addition to *lift*, *assoc*, $deref_v$ and β, the reduction theory includes the following reduction rules:

$$\begin{array}{lll}
(\mu_{ap}) & (\mu\alpha.J)M & \to \mu\alpha.J[\alpha \leftarrow [\alpha](\Box\, M)] \\
(\mu_{let}) & \mathsf{let}\ x = \mu\alpha.J\ \mathsf{in}\ C[x] & \to \mu\alpha.J[\alpha \leftarrow [\alpha](\mathsf{let}\ x = \Box\ \mathsf{in}\ C[x])] \\
(\mu_{lift}) & \mathsf{let}\ x = M_{nv}\ \mathsf{in}\ \mu\alpha.[\beta]N & \to \mu\alpha.[\beta](\mathsf{let}\ x = M_{nv}\ \mathsf{in}\ N) \\
(\mu_{base}) & [\beta]\mu\alpha.J & \to J[\alpha \leftarrow \beta]
\end{array}$$

The relation $\to_{\mu_{need}}$ denotes the compatible closure of $\to$, and $\twoheadrightarrow_{\mu_{need}}$ denotes the reflexive and transitive closure of $\to_{\mu_{need}}$. The weak head standard reduction is defined as follows:

$$\frac{M \to_{\beta,lift,\mu_{ap}} N}{[\alpha]C_{\beta l}[M] \mapsto_{\mu_{need}} [\alpha]C_{\beta l}[N]} \qquad \frac{M \to_{deref_v,assoc,\mu_{let},\mu_{lift}} N}{[\alpha]C_{da}[M] \mapsto_{\mu_{need}} [\alpha]C_{da}[N]} \qquad \frac{J \to_{\mu_{base}} J'}{J \mapsto_{\mu_{need}} J'}$$

The notation $\mapsto\!\!\!\!\twoheadrightarrow_{\mu_{need}}$ stands for the reflexive and transitive closure of $\mapsto_{\mu_{need}}$. Note that we only reduce jumps. A jump J is in weak head normal form if for no $J', J \mapsto_{\mu_{need}} J'$. A weak head normal form (whnf) is a term M such that, either M is $\mu\alpha.J$ with J in weak head normal form or, for no $J, [\alpha]M \mapsto_{\mu_{need}} J$. For example, $\mathsf{let}\ x = \mu\alpha.[\beta]P\ \mathsf{in}\ yx$ is in whnf.

Proposition 12 (Confluence). *$\to_{\mu_{need}}$ is confluent.*

Proposition 13 (Standardization). *Given a term M and whnf N, if $M \twoheadrightarrow_{\mu_{need}} N$ then there exists a whnf N' such that $M \mapsto\!\!\!\!\twoheadrightarrow_{\mu_{need}} N'$ and $N' \twoheadrightarrow_{\mu_{need}} N$.*

The translation between classical call-by-need in natural deduction and sequent calculus form is modified in the following way to cover the classical constructs:

$$\begin{array}{llll}
(\mu\alpha.J)^{\triangleright} & = \mu\alpha.J^{\triangleright} & ([\alpha]M)^{\triangleright} & = M^{\triangleright}_{\alpha} \\
\alpha_{\alpha} & = \alpha & F_{\alpha} & = \Box \\
\alpha_{v\cdot E} & = \alpha_E & F_{v\cdot E} & = F_E[\Box v^{\triangleleft}] \\
\alpha_{\tilde{\mu}x.\langle v \| e\rangle} & = \alpha_e & F_{\tilde{\mu}x.\langle v \| e\rangle} & = \mathsf{let}\ x = \Box\ \mathsf{in}\ F_e[v^{\triangleleft}] \\
(\mu\alpha.c)^{\triangleleft} & = \mu\alpha.c^{\triangleleft} & e^{\triangleleft} & = [\alpha_e]F_e
\end{array}$$

Proposition 14 (Equational correspondence). *Classical call-by-need in natural deduction and sequent calculus form are in equational correspondence:*
(i) $M =_{\mu_{need}} M^{\triangleright\triangleleft}$
(ii) $c =_{lv} c^{\triangleleft\triangleright}$
(iii) If $M =_{\mu_{need}} N$ then $M^{\triangleright} =_{lv} N^{\triangleright}$
(iv) If $c =_{lv} c'$ then $c^{\triangleleft} =_{\mu_{need}} c'^{\triangleleft}$

Notice that the main reason for having only equational correspondence instead of a more precise reduction correspondence is the fact that, in $\lambda\mu_{need}$, μ_{ap} can be applied atomically $(\mu\alpha.J)N_1 \dots N_n \to_{\mu_{ap}} (\mu\alpha.J[\alpha \leftarrow [\alpha]\Box N_1])N_2 \dots N_n$ while in $\overline{\lambda}_{lv}$ the whole applicative context $\Box N_1 \dots N_n$ is moved at once. In particular, the following holds: if $c \to_{lv} c'$ then $c^{\triangleleft} \twoheadrightarrow_{\mu_{need}} c'^{\triangleleft}$.

7 Dual classical call-by-need in sequent calculus form ($\overline{\lambda}_{ln}$)

In call-by-need, the focus is on the consumer and goes to the producer on a need basis. This suggests a dual call-by-need which corresponds to focusing on the producer and going to the consumer on a need basis. To that end, we first extend the classical call-by-need calculus of the previous section, $\overline{\lambda}_{lv}$, with the dual of the implication, the subtraction connective, and then build the dual classical call-by-need calculus by using duality constructions typical from $\overline{\lambda}\mu\tilde{\mu}$-calculi.

While μ and $\tilde{\mu}$ constructs are dual of each other, implicative constructions $\lambda x.t$ and $v \cdot E$ currently have no dual in $\overline{\lambda}_{lv}$. We extend $\overline{\lambda}_{lv}$ by adding constructions for the subtraction connective [10]. Subtraction was already considered in the setting of $\overline{\lambda}\mu\tilde{\mu}$ in Curien *et al.* [11]. We follow the notation introduced by Herbelin in his habilitation thesis [24]. Terms are extended with the construction $v - e$ and contexts with $\tilde{\lambda}\alpha.e$. The corresponding reduction is:

$$(-) \qquad \langle v - e \| \tilde{\lambda}\alpha.e' \rangle \to \langle \mu\alpha.\langle v \| e' \rangle \| e \rangle$$

We can now present the classical call-by-need calculus extended with subtraction, $\overline{\lambda}^{-}_{lv}$. The structural rules are obtained by instantiating $\mathcal{V}$ and $\mathcal{E}$ as:

$$V^-_v = x \mid \lambda x.t \mid (V^-_v - e)$$

$$E^-_l = F^- \mid \tilde{\mu}x.C^{\tilde{\mu}}_l[\langle x \| F^- \rangle] \text{ with } F^- = \alpha \mid v \cdot E^-_l \mid \tilde{\lambda}\alpha.e$$

The syntax for the language with subtraction is finally as follows (with $c = \langle v \| e \rangle$):

command	$c ::= \langle t \| e \rangle$
term	$v ::= V \mid \mu\alpha.c$
linear term	$V ::= x \mid \lambda x.v \mid V - e$
context	$e ::= E \mid \tilde{\mu}x.c$
yielding context	$E ::= F \mid \tilde{\mu}x.C[\langle x \| F \rangle]$
linear context	$F ::= \alpha \mid v \cdot E \mid \tilde{\lambda}\alpha.e$
meta-context	$C ::= \Box \mid \langle \mu\alpha.c \| \tilde{\mu}x.C \rangle$

Using the duality principles developed in [11], we obtain $\overline{\lambda}^{-}_{ln}$ by dualizing $\overline{\lambda}^{-}_{lv}$: The syntax of the calculus is obtained by dualizing $\overline{\lambda}^{-}_{lv}$ syntax and its reductions are also obtained by duality: (β) and $(-)$ are dual of each other while μ_l and $\tilde{\mu}_v$ are respectively turned into:

– the μ-reduction associated with set $E^-_n ::= \alpha \mid v \cdot E^-_n \mid \tilde{\lambda}\alpha.e$, written μ_n

– the $\tilde{\mu}$-reduction associated with set $V_l^- ::= W \mid \mu\alpha.C_l^\mu[\langle W \| \alpha \rangle]$, with $W ::= x \mid \lambda x.t \mid V - e$ (and C_l^μ being the dual of $C_l^{\tilde{\mu}}$), written $\tilde{\mu}_l$.

Since only linear contexts are substituted for context variables, as in call-by-name, but only on a needed basis, we call the resulting calculus lazy call-by-name. Its syntax is given as follows:

$$\begin{array}{lll}
\textit{command} & c & ::= \langle t \| e \rangle \\
\textit{term} & v & ::= V \mid \mu\alpha.c \\
\textit{yielding term} & V & ::= W \mid \mu\alpha.C[\langle W \| \alpha \rangle] \\
\textit{linear term} & W & ::= x \mid \lambda x.v \mid V - e \\
\textit{context} & e & ::= E \mid \tilde{\mu}x.c \\
\textit{linear context} & E & ::= \alpha \mid v \cdot E \mid \tilde{\lambda}\alpha.e \\
\textit{meta-context} & C & ::= \Box \mid \langle \mu\alpha.C \| \tilde{\mu}x.c \rangle
\end{array}$$

The four theories can be discriminated by the following command:

$$\begin{array}{l}
c \equiv \langle\, \mu\alpha.\langle \lambda x.\mu_.\langle \lambda y.x \| \alpha \rangle \| \alpha \rangle \\
\qquad \| \tilde{\mu} f.\langle\, \mu\beta.\langle f \| t \cdot \beta \rangle \\
\qquad\qquad \| \tilde{\mu} x_1.\langle\, \mu\gamma.\langle f \| s \cdot \gamma \rangle \\
\qquad\qquad\qquad \| \tilde{\mu} x_2.\langle x_1 \| x_2 \cdot x_2 \cdot \delta \rangle \rangle \rangle \rangle
\end{array}$$

We call c_1 the command obtained by instantiating t and s to $\lambda x.\lambda y.x$ and $\lambda x.\lambda y.y$, respectively. Then c_1 evaluates to $\langle \lambda x.\lambda y.x \| \delta \rangle$ in lazy call-by-value and to $\langle \lambda x.\lambda y.y \| \delta \rangle$ in call-by-name. We call c_2 the command obtained by instantiating t and s to $\lambda f.\lambda x.\mu\alpha.\langle f \| x \cdot \alpha \rangle$ and $\lambda x.x$. We now consider c_3 to be $\langle \mu\gamma.c_2 \| \tilde{\mu} w.c_1 \rangle$, where w does not occur free in c_1 and γ does no occur free in c_2. In call-by-name and lazy call-by-value, c_3 evaluates as c_1, up to garbage collection. However, c_3 evaluates to $\langle \lambda f.\lambda x.\mu\alpha.\langle f \| x \cdot \alpha \rangle \| \delta \rangle$ in call-by-value, and to $\langle I \| \delta \rangle$ in lazy call-by-name, up to garbage collection. This can be generalized by the following example, where we assume that α_1 does not occur free in c and V, and that x_1 does not occur free in c' and E. If we define

$$c_0 \triangleq \langle \mu\alpha_1.\langle \mu\alpha_2.\langle V \| \alpha_2 \rangle \| \tilde{\mu} y.c \rangle \| \tilde{\mu} x_1.\langle \mu\beta.c' \| \tilde{\mu} x_2.\langle x_2 \| E \rangle \rangle \rangle$$

$$\begin{array}{lll}
\text{then} & c_0 \twoheadrightarrow_n & c'[\beta \leftarrow E[x_2 \leftarrow \mu\beta.c']] \\
& c_0 \twoheadrightarrow_v & c[y \leftarrow V[\alpha_2 \leftarrow \tilde{\mu} y.c]] \\
& c_0 \twoheadrightarrow_{ln} & \langle \mu\alpha_1.c[y \leftarrow \mu\alpha_2.\langle V \| \alpha_2 \rangle] \| \tilde{\mu} x_1.\langle \mu\beta.c' \| \tilde{\mu} x_2.\langle x_2 \| E \rangle \rangle \rangle \\
& c_0 \twoheadrightarrow_{lv} & \langle \mu\alpha_1.\langle \mu\alpha_2.\langle V \| \alpha_2 \rangle \| \tilde{\mu} y.c \rangle \| \tilde{\mu} x_1.c'[\beta \leftarrow \tilde{\mu} x_2.\langle x_2 \| E \rangle] \rangle
\end{array}$$

8 Conclusions and Future work

The advantage of studying evaluation order in the context of sequent calculus has shown its benefits: extending the calculus (both syntax and reduction theory) to the classical case simply corresponds to going from one context variable to many. The study has also suggested how to provide a call-by-need version of Parigot's $\lambda\mu$-calculus, and in the minimal case, has led to a new notion of

standard reduction, which applies the *lift* and *assoc* rule eagerly. In the minimal case, the single context variable, called $\star$, could be seen as the constant $\mathbf{tp}$ discussed in [6,5]. In the cited work, it is also presented how delimited control can be captured by extending $\mathbf{tp}$ to a dynamic variable named $\widehat{tp}$. This suggests that one could use $\widehat{tp}$ instead of $\mathbf{tp}$ to represent computations also in the minimal setting. Since evaluation goes under a $\widehat{tp}$, it means that one would obtain a different notion of standard reduction, which would correspond to the one of Ariola *et al.* [3] and Maraist *et al.* [27].

A benefit of sequent calculus over natural deduction in both call-by-name and call-by-value is that the standard redex in the sequent calculus always occurs at the top of the command. In other words, there is no need to perform an unbounded search to reach the standard redex [2]: this search is embedded in the structural reduction rules. However, this does not apply to our call-by-need sequent calculus: the standard redex can be buried under an arbitrary number of bindings. This can be easily solved by considering a calculus with explicit substitutions. A command now becomes $\langle v \| e \rangle \boldsymbol{\tau}$, where $\boldsymbol{\tau}$ is a list of declarations. For example, the critical pair will be solved as: $\langle \mu\alpha.c \| \tilde{\mu}x.c' \rangle \boldsymbol{\tau} \to c'[x = \mu\alpha.c]\boldsymbol{\tau}$ and the switching of context is realized by the rule: $\langle x \| E \rangle \boldsymbol{\tau_0}[x := \mu\alpha.c]\boldsymbol{\tau_1} \to c[\alpha := \tilde{\mu}x.\langle x \| E \rangle \boldsymbol{\tau_0}]\boldsymbol{\tau_1}$. This will naturally lead us to developing abstract machines, which will be compared to the abstract machines of Garcia *et al.* [22] and Danvy *et al.* [15], inspired by natural deduction.

We have related the lazy call-by-value with subtraction to its dual. We plan to provide a simulation of lazy call-by-value in lazy call-by-name and vice-versa, without the use of subtraction. We are also interested in devising a complete set of axioms with respect to a classical extension of the call-by-need continuation-passing style of Okasaki *et al.* [30]. A natural development will then be to extend our lazy call-by-value and lazy call-by-name with delimited control. Following a suggestion by Danvy, we will investigate connections between our lazy call-by-name calculus and a calculus with futures [29]. At last, we want to better understand the underlying logic or type system.

Aknowledgements: The authors wish to thank Olivier Danvy and the anonynmous referees for valuable comments and suggestions. Zena M. Ariola has been supported by NSF grant CCF-0917329. This research has been developed under INRIA Équipe Associée SEMACODE.

References

1. Z. Ariola and H. Herbelin. Minimal classical logic and control operators. In *ICALP'03*, volume 2719 of *Lecture Notes in Computer Science*. Springer, 2003.
2. Z. M. Ariola, A. Bohannon, and A. Sabry. Sequent calculi and abstract machines. *Transactions on Programming Languages and Systems (TOPLAS)*, 31(4):275–317, 2009.
3. Z. M. Ariola and M. Felleisen. The call-by-need lambda calculus. *J. Funct. Program.*, 7(3):265–301, 1997.

4. Z. M. Ariola and H. Herbelin. Control reduction theories: the benefit of structural substitution. *J. Funct. Program.*, 18(3):373–419, 2008.
5. Z. M. Ariola, H. Herbelin, and A. Sabry. A proof-theoretic foundation of abortive continuations. *Higher Order Symbol. Comput.*, 20:403–429, December 2007.
6. Z. M. Ariola, H. Herbelin, and A. Sabry. A type-theoretic foundation of delimited continuations. *Higher Order Symbol. Comput.*, 22:233–273, September 2009.
7. H. Barendregt and S. Ghilezan. Lambda terms for natural deduction, sequent calculus and cut elimination. *J. Funct. Program.*, 10:121–134, January 2000.
8. H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North Holland, Amsterdam, 1984.
9. A. Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 2:33, 346–366, 1932.
10. T. Crolard. Subtractive Logic. *Theoretical Computer Science*, 254(1–2):151–185, 2001.
11. P.-L. Curien and H. Herbelin. The duality of computation. In *International Conference on Functional Programming*, pages 233–243, 2000.
12. V. Danos, J.-B. Joinet, and H. Schellinx. LKQ and LKT: sequent calculi for second order logic based upon dual linear decompositions of the classical implication. In *Advances in Linear Logic*, volume 222, pages 211–224. Cambridge University Press, 1995.
13. V. Danos, J.-B. Joinet, and H. Schellinx. A new deconstructive logic: Linear logic. *J. Symb. Log.*, 62(3):755–807, 1997.
14. O. Danvy, J. Johannsen, and I. Zerny. A walk in the semantic park. In *PEPM11*, New York, NY, USA, 2011. ACM.
15. O. Danvy, K. Millikin, J. Munk, and I. Zerny. Defuntionalized interpreters for call-by-need evaluation. In *Functional and Logic Programming, FLOPS2010*, 2010.
16. R. David and W. Py. Lambda-mu-calculus and Böhm's theorem. *J. Symb. Log.*, 66(1):407–413, 2001.
17. J. H. Fasel, P. Hudak, S. Peyton Jones, and P. W. (editors). Haskell special issue. *SIGPLAN Notices*, 27(5), May 1992.
18. M. Felleisen and D. Friedman. Control operators, the secd machine, and the lambda-calculus. In *Formal description of programming concepts-III*, pages 193–217. North-Holland, 1986.
19. M. Felleisen, D. Friedman, and E. Kohlbecker. A syntactic theory of sequential control. *Theoretical Computer Science*, 52(3):205–237, 1987.
20. M. Felleisen, D. Friedman, E. Kohlbecker, and B. Duba. Reasoning with continuations. In *First Symposium on Logic and Computer Science*, pages 131–141, 1986.
21. A. Filinski. *Declarative Continuations and Categorical Duality*. Master thesis, DIKU, Danmark, Aug. 1989.
22. R. Garcia, A. Lumsdaine, and A. Sabry. Lazy evaluation and delimited control. In *POPL '09: Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 153–164, New York, NY, USA, 2009. ACM.
23. G. Gentzen. Investigations into logical deduction. In M. Szabo, editor, *Collected papers of Gerhard Gentzen*, pages 68–131. North-Holland, 1969.
24. H. Herbelin. C'est maintenant qu'on calcule. In *Habilitation à diriger les reserches*, 2005.
25. H. Herbelin and S. Zimmermann. An operational account of call-by-value minimal and classical λ-calculus in "natural deduction" form. In P.-L. Curien, editor,

Ninth International Conference, TLCA '07, Brasilia, Brazil. July 2009, Proceedings, volume 5608 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2009.

26. Y. Lafont, B. Reus, and T. Streicher. Continuations semantics or expressing implication by negation. Technical Report 9321, Ludwig-Maximilians-UniversitÃd't, MÃijnchen, 1993.
27. J. Maraist, M. Odersky, and P. Wadler. The call-by-need λ-calculus. *J. Funct. Program.*, 8(3):275–317, 1998.
28. E. Moggi. Computational λ-calculus and monads. In *Logic in Computer Science*, 1989.
29. J. Niehren, J. Schwinghammer, and G. Smolka. A concurrent lambda calculus with futures. *Theor. Comput. Sci.*, 364:338–356, November 2006.
30. C. Okasaki, P. Lee, and D. Tarditi. Call-by-need and continuation-passing style. In *Lisp and Symbolic Computation*, pages 57–81. Kluwer Academic Publishers, 1993.
31. M. Parigot. Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction. In *LPAR 92*, pages 190–201. Springer-Verlag, 1992.
32. G. D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theoretical Comput. Sci.*, 1:125–159, 1975.
33. D. Prawitz. *Natural Deduction, a Proof-Theoretical Study*. Almquist and Wiksell, Stockholm, 1965.
34. S. Ronchi Della Rocca and L. Paolini. *The Parametric λ-Calculus: a Metamodel for Computation*. Texts in Theoretical Computer Science: An EATCS Series. Springer-Verlag, 2004.
35. A. Sabry and M. Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6(3-4):289–360, 1993.
36. A. Saurin. Separation with streams in the $\lambda\mu$-calculus. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 356–365. IEEE Computer Society, 2005.
37. P. Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Mathematical Structures in Computer Science*, 11(2):207–260, 2001.

9 Abstract Machines for Call-by-need

9.1 An Abstract Machine for $\overline{\lambda}_{lv}$

An abstract machine realizes the standard reduction in a tail recursive manner. To that end, a command is paired with an ordered list of declarations for regular and continuation variables. We directly present the abstract machine for the classical case, and then discuss how to restrict it to the minimal case. We assume that $\boldsymbol{\tau_0}$ does not contain a binding for either x or α.

$$
\begin{array}{lll}
\text{State} & st ::= c\boldsymbol{\tau} & \\
\text{Binding} & \tau ::= [x{=}t] \mid [\alpha{=}F] \mid [\alpha = \tilde{\mu}x.\langle x \| F\rangle \boldsymbol{\tau}] & \\
\text{Binding vector} & \boldsymbol{\tau} ::= \epsilon \mid \tau\boldsymbol{\tau} &
\end{array}
$$

$$
\begin{array}{lll}
\langle t \| \tilde{\mu}x.c\rangle \boldsymbol{\tau} & \rightsquigarrow_{lv} & c[x{=}t]\boldsymbol{\tau} \\
\langle \mu\alpha.c \| F\rangle \boldsymbol{\tau} & \rightsquigarrow_{lv} & c[\alpha{=}F]\boldsymbol{\tau} \\
\langle x \| t \cdot E\rangle \boldsymbol{\tau} & \rightsquigarrow_{lv} & \begin{cases} \langle V \| t \cdot E\rangle \boldsymbol{\tau} & \text{if } \boldsymbol{\tau}{=}\boldsymbol{\tau_0}[x{=}V]\boldsymbol{\tau_1} \\ c[\alpha{=}\tilde{\mu}x.\langle x \| t \cdot E\rangle \boldsymbol{\tau_0}]\boldsymbol{\tau_1} & \text{if } \boldsymbol{\tau}{=}\boldsymbol{\tau_0}[x{=}\mu\alpha.c]\boldsymbol{\tau_1} \end{cases} \\
\langle x \| \alpha\rangle \boldsymbol{\tau} & \rightsquigarrow_{lv} & \begin{cases} \langle V \| \alpha\rangle \boldsymbol{\tau} & \text{if } \boldsymbol{\tau}{=}\boldsymbol{\tau_0}[x{=}V]\boldsymbol{\tau_1} \\ \langle x \| F\rangle \boldsymbol{\tau'}[y{=}x]\boldsymbol{\tau} & \text{if } \boldsymbol{\tau}{=}\boldsymbol{\tau_0}[\alpha{=}\tilde{\mu}y.\langle y \| F\rangle \boldsymbol{\tau'}]\boldsymbol{\tau_1} \wedge [x{=}V] \notin \tau \\ c[\beta{=}\tilde{\mu}x.\langle x \| \beta\rangle \boldsymbol{\tau_0}]\boldsymbol{\tau_1} & \text{if } \boldsymbol{\tau}{=}\boldsymbol{\tau_0}[x{=}\mu\beta.c]\boldsymbol{\tau_1} \wedge [\alpha{=}\tilde{\mu}y.\langle y \| F\rangle \tau'] \notin \tau \end{cases} \\
\langle \lambda x.t \| s \cdot E\rangle \boldsymbol{\tau} & \rightsquigarrow_{lv} & \langle s \| \tilde{\mu}x.\langle t \| E\rangle\rangle \boldsymbol{\tau} \\
\langle \lambda x.t \| \alpha\rangle \boldsymbol{\tau} & \rightsquigarrow_{lv} & \begin{cases} \langle \lambda x.t \| F\rangle \boldsymbol{\tau'}[y{=}\lambda x.t]\boldsymbol{\tau} & \text{if } \boldsymbol{\tau}{=}\boldsymbol{\tau_0}[\alpha{=}\tilde{\mu}y.\langle y \| F\rangle \boldsymbol{\tau'}]\boldsymbol{\tau_1} \\ \langle \lambda x.t \| F\rangle \boldsymbol{\tau} & \text{if } \boldsymbol{\tau}{=}\boldsymbol{\tau_0}[\alpha{=}F]\boldsymbol{\tau'} \end{cases}
\end{array}
$$

The first two cases of the abstract machine are straightforward. If a variable, which is surrounded by an applicative context, is bound to a value then that value is substituted. If the variable is bound to a computation, a switching of context occurs. To understand the next case, consider the state $\langle x \| \alpha\rangle \boldsymbol{\tau}$, where $\boldsymbol{\tau}$ is $[x{=}V][\alpha = \tilde{\mu}y.\langle y \| F\rangle \epsilon]$, we need to make sure that the substitution for x occurs before the substitution for α, that is, the next state should be $\langle V \| \alpha\rangle \boldsymbol{\tau}$ and not $\langle x \| F\rangle [y{=}x]\boldsymbol{\tau}$. That explains why we first check if x is bound to a value. Consider now state $\langle x \| \alpha\rangle [x{=}\mu\beta.c][\alpha{=}\tilde{\mu}y.\langle y \| F\rangle]$, we need to make sure that the substitution for y occurs before switching to c. Thus, the next state should be $\langle x \| F\rangle [y = x][x{=}\mu\beta.c][\alpha{=}\tilde{\mu}y.\langle y \| F\rangle]$.

States of the abstract machine are translated into the $\overline{\lambda}_{lv}$ calculus as follows:

$$
\begin{array}{ll}
(c\boldsymbol{\tau})^{\circ} & = \boldsymbol{\tau}^{\circ}[c] \\
\epsilon^{\circ} & = \Box \\
([x = \mu\alpha.c_1]\boldsymbol{\tau})^{\circ} & = \boldsymbol{\tau}^{\circ}[\langle \mu\alpha.c_1 \| \tilde{\mu}x.\Box\rangle] \\
([x = V]\boldsymbol{\tau})^{\circ} & = \boldsymbol{\tau}^{\circ}[\Box[x \leftarrow V]] \\
([\alpha = F]\boldsymbol{\tau})^{\circ} & = \boldsymbol{\tau}^{\circ}[\Box[\alpha \leftarrow F]] \\
([\alpha = \tilde{\mu}y.\langle y \| F\rangle \boldsymbol{\tau'}]\boldsymbol{\tau})^{\circ} & = \boldsymbol{\tau}^{\circ}[\Box[\alpha \leftarrow \tilde{\mu}y.\boldsymbol{\tau'}^{\circ}[\langle y \| F\rangle]]]
\end{array}
$$

(i) if $st \rightsquigarrow_{lv} st'$ then $(st)^{\circ} \mapsto^{0/1} (st')^{\circ}$;

(ii) if $c \mapsto\!\!\!\!\rightarrow_{lv} c'$ then, for any binding $\boldsymbol{\tau}$, there exists a state $(c'\ \boldsymbol{\tau_0})$ such that $c\ \boldsymbol{\tau} \mapsto\!\!\!\!\rightarrow_{lv} c''\ \boldsymbol{\tau_0}\boldsymbol{\tau}$ and $c' \equiv (c''\ \boldsymbol{\tau_0})^{\circ}$.

Proof. (i) Follows from the fact that the translation of $c\boldsymbol{\tau}$ is of the form $C_{lv}[c\sigma]$ for some meta-context C_{lv} and some substitution σ.

More precisely, the first rule corresponds to an identity in case x is bound to a computation, otherwise it corresponds to $\tilde{\mu}_v$. The second rule corresponds to a β step; The third rule corresponds to a μ_l step while the fourth rule is an identity. The fifth rule corresponds to a μ_l step:

$$\boldsymbol{\tau_1}^{\circ}[\langle \mu\alpha.c \| \tilde{\mu}x.\boldsymbol{\tau_0}^{\circ}[\langle x \| t \cdot E\rangle]\rangle] \mapsto \boldsymbol{\tau_1}^{\circ}[c[\alpha \leftarrow \tilde{\mu}x.\boldsymbol{\tau_0}^{\circ}[\langle x \| t \cdot E\rangle]]]$$

The sixth and seventh rules correspond to a $\tilde{\mu}_v$ step (below, the case of the seventh rule):

$$\boldsymbol{\tau}^\circ[\boldsymbol{\tau_0}^\circ[\langle \lambda x.t \| \tilde{\mu} y.\boldsymbol{\tau'}^\circ[\langle y \| F\rangle]\rangle]] \mapsto \boldsymbol{\tau}^\circ[\boldsymbol{\tau_0}^\circ[\boldsymbol{\tau'}^\circ[\langle \lambda x.t \| F\rangle][y \leftarrow \lambda x.t]]]$$

The last rule corresponds to an identity. ∎

The restriction to the minimal case is easily obtained by remembering that once the single continuation variable $\star$ is used, there is no need to keep its binding. The classical case has to consider a state such as $\langle \lambda x.\mu\delta.\langle z \| \alpha\rangle \| \alpha\rangle[\alpha = \tilde{\mu} x.\langle x \| F\rangle \epsilon]$, therefore, after having substituted for α its binding needs to be mantained in place. Instead, in the minimal case, δ and α have to be the same, therefore, one would have: $\langle \lambda x.\mu\star.\langle z \| \star\rangle \| \star\rangle[\star = \tilde{\mu} x.\langle x \| F\rangle \epsilon]$, once the substitution occurs the binding is deleted. This is captured by the rules below:

$$\begin{array}{lll}
\langle x \| \star\rangle \boldsymbol{\tau} & \rightsquigarrow_{mlv} \langle x \| F\rangle \boldsymbol{\tau'}[y = x]\boldsymbol{\tau_0}\boldsymbol{\tau_1} & \text{if } \boldsymbol{\tau} = \boldsymbol{\tau_0}[\star = \tilde{\mu} y.\langle y \| F\rangle \boldsymbol{\tau'}]\boldsymbol{\tau_1} \wedge [x = V] \notin \tau \\
\langle \lambda x.t \| \star\rangle \boldsymbol{\tau} & \rightsquigarrow_{mlv} \begin{cases} \langle \lambda x.t \| F\rangle \boldsymbol{\tau'}[y = \lambda x.t]\boldsymbol{\tau_0}\boldsymbol{\tau_1} & \text{if } \boldsymbol{\tau} = \boldsymbol{\tau_0}[\star = \tilde{\mu} y.\langle y \| F\rangle \boldsymbol{\tau'}]\boldsymbol{\tau_1} \\ \langle \lambda x.t \| F\rangle \boldsymbol{\tau_0}\boldsymbol{\tau'} & \text{if } \boldsymbol{\tau} = \boldsymbol{\tau_0}[\star = F]\boldsymbol{\tau'} \end{cases}
\end{array}$$

The minimal restriction differs from the abstract machine of Garcia *et al.* [22], since the applicative context is kept separate from the binding context. This has the advantage that once a value is reached, the binding context does not need to be recognized and collected up to the nearest applicative context. This avoids copying and re-installing of bindings, as shown below. The execution of $(\lambda x_1.\lambda x_2.\lambda x_3.x_2 t) t_1 t_2 t_3$ (for simplicity we do not show the set X used to generate unique names) according to Garcia *et al.* is :

$$\begin{array}{ll}
\langle \Box, (\lambda x_1.\lambda x_2.\lambda x_3.x_2 t) t_1 t_2 t_3\rangle & \mapsto\!\!\!\!\mapsto \\
\langle \Box t_3 \circ (\lambda x_1.\Box) t_1 \circ (\lambda x_2.\Box) t_2, \lambda x_3.x_2 t\rangle & \mapsto \text{ Bindings are copied} \\
\langle \Box, [\![(\lambda x_1.\Box) t_1 \circ (\lambda x_2.\Box) t_2, \lambda x_3.x_2 t]\!] t_3\rangle & \mapsto \text{ Bindings are re-installed} \\
\langle (\lambda x_1.\Box) t_1 \circ (\lambda x_2.\Box) t_2 \circ (\lambda x_3.\Box) t_3, x_2 t\rangle &
\end{array}$$

Our execution is:

$$\begin{array}{ll}
\langle \lambda x_1.\lambda x_2.\lambda x_3.\mu\star.\langle x_2 \| t \cdot \star\rangle \| t_1 \cdot t_2 \cdot t_3 \cdot \star\rangle & \rightsquigarrow\!\!\!\!\rightsquigarrow_{mlv} \\
\langle \lambda x_3.\mu\star.\langle x_2 \| t \cdot \star\rangle \| t_3 \cdot \star\rangle[x_2 = t_2][x_1 = t_1] & \rightsquigarrow\!\!\!\!\rightsquigarrow_{mlv} \\
\langle \mu\star.\langle x_2 \| t \cdot \star\rangle \| \star\rangle[x_3 = t_3][x_2 = t_2][x_1 = t_1] &
\end{array}$$

This separation between the applicative context and the binding context is essential in the classical case, since the binding contexts should not be part of the captured context, as described above in Section 6.

9.2 An Abstract machine for $\boldsymbol{\lambda_{need}}$

The following is wrong

Remark 5 The abstract machine for the natural deduction system is the same as the abstract machines given by Garcia *et al.* [22] and Danvy *et al.* [15] with one important difference. As stated in [14], an abstract machine corresponds to

the defunctionalization of the standard reduction. However, that does not apply to these abstract machines. There is a mismatch between the standard reduction and the steps of the abstract machine. For example, in the term $\mathsf{let}\ z = (\mathsf{let}\ x = V\ \mathsf{in}\ x)\ \mathsf{in}\ z$, the machine would perform the *assoc* redex, whereas the standard reduction given in [3] and [27] would perform the *deref* redex.