# call-by-name, call-by-value and abstract machines

*Author:*
Remy Viehoff

*Supervisor:*
Prof. Dr. Herman Geuvers

*Date:*
June, 2012

*Thesis number:*
663

# Contents

# Chapter 1

# Introduction

The Curry-Howard isomorphism is a remarkable correspondence between logic on the one side and $\lambda$-calculi on the other side. Under the isomorphism, formulas are mapped to types, proofs to terms and proof normalization steps correspond to reduction.

From a logical perspective, the isomorphism allows us to use terms from a $\lambda$-calculus as a short way to write down proofs. But, since $\lambda$-calculi are also used as a foundation for programming languages, the terms in such calculus can be interpreted as programs. This provides us with a means to talk about "the computational content" of a proof.

From a computational perspective, the isomorphism allows us to assign logical formulae to terms in the $\lambda$-calculus. These formulae can then be interpreted as the specification of the program (term) it is assigned to and, hence, we can prove that a program implements its specification. In some cases the isomorphism even allows us to extract a program from the proof of a logical formula. This process is known as *program extraction*.

For quite some time it was thought that only intuitionistic logics could be related to $\lambda$-calculi i.e. that only proofs in intuitionistic logics had computational content. It is was Griffin's pioneering work [Gri89] that showed that Felleisen and Hieb's control operator $\mathcal{C}$ [FH92] could be typed with the double negation elimination rule and, hence, that proofs in a classical logic could have computational content. On the computational level, the control operator $\mathcal{C}$ can be seen as an exception handling mechanism similar to Scheme's *call/cc* (call-with-current-continuation) construct.

Since Griffin's extension of the Curry-Howard isomorphism to classical logic, there has been a lot of work on combining various classical logics with theories of control. In [Her10], Herbelin investigates the intuitionistic quantifier calculus (IQC) extended with Markov's principle, which corresponds to a catch and throw control mechanism at the computational level. In [Par92], Parigot augments the $\lambda$-calculus with a control-like operator called $\mu$. At the logical level, Parigot has a multi conclusion variant of natural deduction which can prove Peirce's law. These are just two examples of work that has been done in this area, but there are many more e.g. [Cro99, RS94, Nak92, Her95, CH00].

In this thesis we focus mainly on the computational side of the Curry-Howard isomorphism. On that side, we have $\lambda$-calculi that can be seen as stripped-down versions of functional programming languages, which make them ideal for

investigating some of the properties and dualities of programming languages such as input/output and program/context. In [CH00] Herbelin and Curien introduce the $\bar{\lambda}\mu\tilde{\mu}$-calculus, which shows that there is also a duality between the reduction systems *call-by-name* and *call-by-value* (also see [Fil89, Sel01]): call-by-name does not evaluate arguments before copying them into the body of a function, while call-by-value only copies arguments into the body of a function if they are first evaluated to a value. In $\bar{\lambda}\mu\tilde{\mu}$, the call-by-name and call-by-value reduction systems live side by side. By resolving the non-determinism in its reduction system we obtain one of those specific systems.

Programming languages, by themselves, are, however, not very useful unless we define some sort of machine that can compute/evaluate a program in that language. In the case of a $\lambda$-calculus, we can define an abstract machine that does this for us (see e.g. [Lan64, Plo75, Kri07, DG98]). Such an abstract machine is defined in terms of reductions that manipulate the state of an environment and/or a stack. The machine reductions are usually deterministic and mimic the weak head reduction strategy that is defined in terms of the reductions in the $\lambda$-calculus.

In [CH00], Herbelin and Curien define an abstract machine for the $\bar{\lambda}\mu\tilde{\mu}$-calculus. However, they do not provide a weak head reduction strategy for the calculus and do not show that the machine is correct and complete for this strategy. This is what the main contribution of this thesis is: we first define a weak head reduction strategy for $\bar{\lambda}\mu$, which is a restriction of $\bar{\lambda}\mu\tilde{\mu}$. We then show that the restriction of Herbelin an Curiens machine to $\bar{\lambda}\mu$ is correct and complete for this strategy. We present a similar result for the full $\bar{\lambda}\mu\tilde{\mu}$ machine and the call-by-name and call-by-value specific versions of that machine.

## 1.1 Exception handling

Many programming languages contain some sort of mechanism that enables a programmer to handle exceptions that occur during a computation. Such mechanisms, which are commonly referred to as *exception handling* or *control* mechanisms, allow a programmer to clearly distinguish "normal" behaviour from "exceptional" behaviour. The canonical example is that of the function *assoc*, whose intention is to find the value associated to the label $x$ in *lst*:

```
let rec assoc lst x =
  match lst with
  | []           -> ExceptionalValue
  | (y, v) :: ys -> if y = x
                    then v
                    else (assoc ys x)
```

Indeed, in the case where *lst* is empty or does not contain the label $x$, we cannot return the value associated to $x$ since it simply does not exist. However, we must still return some value. A solution could be to return some fixed value denoting this erroneous case. A caller of this function must then check, each time *assoc* is called, whether it returns the "erroneous value" or a "normal value". A nicer solution can be given using Lisp's *control operators* **catch** and **throw**.

The intuition behind the notation **catch** $\alpha$ $p$ is to first evaluate $p$. If evaluation of $p$ yields a *normal* result $r$, then **catch** $\alpha$ $p$ yields $r$. If, during evaluation

of $p$, we encounter an *exception* **throw** $\alpha$ $e$, then **catch** $\alpha$ $p$ yields $e$. Consider the following version of *assoc* in which we have incorporated the control operators:

```
let assoc lst x = catch α (assoc_throw lst x)

let rec assoc_throw lst x =
  match lst with
  | []          -> throw α exit(``Label x not found!'')
  | (y,v) :: ys -> if y == x
                   then v
                   else (assoc ys x)
```

Here, the function *exit* just exits the program and prints the message in its first parameter.

We can use the $\lambda$-calculus to reason formally about functional programs. While the $\lambda$-calculus itself does not support exception handling, there exist extensions of the calculus that do incorporate exception handling mechanisms (such as Herbelin's $\lambda IQC_{MP}$ [Her10]) or provide the means to define one (e.g. Parigot's $\lambda\mu$-calculus [Par92, Kre10]). In this thesis we will show that we can define an exception handling mechanism for $\bar{\lambda}\mu$ and $\bar{\lambda}\mu\tilde{\mu}$ that is similar to the one defined in [Kre10] for $\lambda\mu$.

## 1.2 Overview

The remainder of this thesis is structured as follows:

- In Chapter 2 we introduce some notions that will be used later on in this thesis. We do this by first recalling the simply typed $\lambda$-calculus in Section 2.1 and then Parigot's $\lambda\mu$-calculus in Section 2.2. In both cases we define a weak head reduction strategy and give an abstract machine that can evaluate terms in the calculus using this strategy.

- Our main contributions are included in Chapter 3 and 4. In Chapter 3 we start by discussing the $\bar{\lambda}\mu$-calculus, which was originally introduced by Herbelin in [Her95]. In particular, we show that $\bar{\lambda}\mu$ is isomorphic to Parigot's $\lambda\mu$ by defining the inverse of the map $\mathcal{N}$, which is defined in [CH00], and by showing that these maps are, indeed, each others inverse and preserve reduction and typing.

  In Section 3.2 we show that we can define an exception handling mechanism for $\bar{\lambda}\mu$ that is similar to the one defined in [Kre10] for $\lambda\mu$.

  In Section 2.2.2, we define the weak head reduction strategy for $\bar{\lambda}\mu$ and show that a restriction of the abstract machine defined in [CH00] is correct and complete for this strategy.

- In Chapter 4 we start by introducing the $\bar{\lambda}\mu\tilde{\mu}$-calculus and show that, by restricting its reduction system in two ways, we obtain two confluent reduction systems: one *call-by-name* and one *call-by-value* system.

  In Section 4.1 we show that we can define an exception handling mechanism for $\bar{\lambda}\mu\tilde{\mu}$ that is similar to the one defined in [Kre10] for $\lambda\mu$. We moreover show that we can add a reduction to our exception handling

mechanism and thereby obtain a mechanism similar to that of $\lambda IQC_{MP}$ [Her10], since we can now perform call-by-value reduction steps.

In Section 4.2, we show that the notion of weak head reduction for $\bar{\lambda}\mu$ can be extended to $\bar{\lambda}\mu\tilde{\mu}$ and that Herbelin and Curien's machine [CH00] is correct and complete for this extended notion of weak head reduction. We moreover show that the weak head reduction system can be restricted, just like the reduction system, and that we obtain call-by-name and call-by-value specific strategies. The abstract machine can also be restricted in two ways to obtain a call-by-name and call-by-value specific version, which are correct and complete for their respective notions of weak head reduction.

- We conclude in Chapter 5.

## 1.3 Acknowledgements

I want thank my supervisor Herman Geuvers for introducing me to the world of $\lambda$-calculi and the Curry-Howard isomorphism. I also want to thank him for the numerous hours of discussion and his continuous feedback. Without his supervision, this thesis would surely not have been as it is now.

# Chapter 2

# Background

In this chapter we introduce some notions that will be used further on in this thesis. In Section 2.1 we introduce the simply typed $\lambda$-calculus along with its operational semantics in the form of an abstract machine. In Section 2.2 we introduce the $\lambda\mu$-calculus, which is an extension of the $\lambda$-calculus, and an abstract machine which can be seen as its operational semantics.

## 2.1 The simply typed $\lambda$-calculus

In this section we briefly introduce the simply typed $\lambda$-calculus. For a more elaborate discussion of the calculus we refer to [Chu40, BDS12]. The simply typed lambda calculus ($\lambda_{\to}$) was introduced to cope with some of the problems that the untyped $\lambda$-calculus ($\lambda$) was suffering from. For instance: in $\lambda$, functions work on arbitrary inputs. That is: each functions is, in a sense, polymorphic. In the $\lambda_{\to}$-calculus each function has a fixed domain.

Terms in the simply typed $\lambda$-calculus are typed by formulas from the minimal first-order propositional logic. We also call this set of formulas the *simple types*.

**Definition 1.** The set of minimal first-order propositional formulas is generated by the following grammar, where $X, Y, \ldots$ range over an infinite set of atomic propositions.

$$A, B ::= A \to B \ \mid \ X$$

**Definition 2.** The set $\Lambda$ of $\lambda$-terms is generated by the following grammar, where $x, y, \ldots$ range over the infinite set of term-variables $\mathcal{X}$.

$$M, N ::= x \ \mid \ \lambda x.M \ \mid \ (MN)$$

We let applications associate to the left i.e. $MN_1N_2 \ldots N_k = (((MN_1)N_2)\ldots)N_k$. The construct $\lambda x.M$ *binds* $x$ in $M$. A variable is called a *free* variable if it is not bound. In $\lambda_{\to}$, $\lambda$-abstraction is the only binding construct. The set of free variables of a term is determined by the function $FV$, which is given below. A term

$M$ is called *closed* if $M$ does not contain any free variables i.e. if $FV(M) = \emptyset$.

$$\begin{aligned} FV(x) &= \{x\} \\ FV(\lambda x.M) &= FV(M) - \{x\} \\ FV(MN) &= FV(M) \cup FV(N) \end{aligned}$$

Throughout this thesis we stick to the convention that the names of bound variables are chosen maximally fresh and different from the names of free variables. Moreover, we call two terms $M, N$ $\alpha$-equivalent (notation $M \equiv N$) if they differ only in the names of their bound variables. For example: $\lambda x.\lambda y.x \equiv \lambda k.\lambda l.k$, but $\lambda x.\lambda y.x$ is *not* $\alpha$-equivalent to $\lambda k.\lambda l.l$. We will identify $\alpha$-equivalent terms, that is: we work on $\alpha$ equivalence classes $(\Lambda/\equiv)$ of terms.

The derivation rules that determine whether some term $M$ has type $A$ are included in Figure 2.1. We write $\Gamma \vdash M : A$ and call $M$ a *well-typed* term, if the term $M$ has type $A$ in context $\Gamma$. A *context* is a list of bindings of the form $x : A, y : B, \ldots$, where $x, y, \ldots \in \mathcal{X}$ and $A, B, \ldots$ are formulas.

$$\frac{}{\Gamma, x : A \vdash x : A}\ \text{AX} \qquad \frac{\Gamma, x : A \vdash M : A}{\Gamma \vdash \lambda x.M : A \to B}\ \to_I \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}\ \to_E$$

Figure 2.1: Typing rules for $\lambda_\to$

We interpret the notation $(\lambda x.M) : A \to B$ as the function $\lambda x.M$ with domain $A$ and co-domain $B$, moreover $x$ is an element of the domain $A$. We let $\to$ associate to the right i.e. $A \to B \to C \to D = A \to (B \to (C \to D))$.

**Example 1.** Using the rules in Figure 2.1, we can show that the term $(\lambda x.\lambda y.x)(\lambda z.z)$ has type $B \to A \to A$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : A \to A, y : B \vdash x : A \to A}\ \text{AX}}{x : A \to A \vdash \lambda y.x : B \to A \to A}\ \to_I}{\vdash (\lambda x.\lambda y.x) : (A \to A) \to B \to A \to A}\ \to_I \qquad \dfrac{\dfrac{}{z : A \vdash z : A}\ \text{AX}}{\vdash \lambda z.z : A \to A}\ \to_I}{\vdash (\lambda x.\lambda y.x)(\lambda z.z) : B \to A \to A}\ \to_E$$

In fact, $(\lambda x.\lambda y.x)(\lambda z.z)$ is a closed (well-typed) term of type $B \to A \to A$.

*end of example*

**Definition 3.** Reduction on $\lambda$-terms is defined as the compatible closure of the following rule:

$$(\lambda x.M)N \to_\beta M[N/x]$$

We write $\to^*$ for the reflexive, transitive closure and $\to^+$ for the transitive closure of the reduction in Definition 3. Any term of the form $(\lambda x.M)N$ is called a *redex* and the process of reducing such redex is also called *contraction*. This process always ends, since reduction in $\lambda_\to$ is strongly normalizing. If $N$ contains no more redexes, we say that $N$ is in *normal form* and we write $N \nrightarrow$.

**Lemma 1.** *The reductions in $\lambda_\rightarrow$ are strongly normalizing. That is: the process of reduction always terminates.*

*Proof.* See e.g. [GTL89]. □

**Definition 4.** Substitution $M[N/x]$ is defined as follows:

$$\begin{aligned} x[N/x] &= N \\ y[N/x] &= y \quad (\text{if } y \neq x) \\ (\lambda y.M)[N/x] &= \lambda y.M[N/x] \\ (MN)[P/x] &= (M[P/x])(N[P/x]) \end{aligned}$$

Moreover, substitution is *capture-free*, which means that all bound occurrences of $x$ in $M$ must be renamed to avoid being captured by the substitution. For example:

$$\begin{aligned} &(\lambda x.x)[N/x] \equiv (\lambda y.y)[N/x] = \lambda y.y \equiv \lambda x.x && \text{(Capture free substitution)} \\ &(\lambda x.x)[N/x] = \lambda x.N && \text{(Substitution with capture)} \end{aligned}$$

**Example 2.**

$$\begin{aligned} &(\lambda x.x)((\lambda y.y)(\lambda z.z)) \rightarrow_\beta (\lambda x.x)(\lambda z.z) \rightarrow_\beta \lambda z.z \quad \text{, but also} \\ &(\lambda x.x)((\lambda y.y)(\lambda z.z)) \rightarrow_\beta (\lambda y.y)(\lambda z.z) \rightarrow_\beta \lambda z.z \end{aligned}$$

*end of example*

Remember, from the introduction, that the type of a program (term) can be seen as the specification of that program. Of course, we want this specification to be preserved under reduction; we do not want a program to change into an entirely different program just by reducing it. To that end, we check that the reduction(s) satisfy subject reduction. But first we need the following substitution lemma.

**Lemma 2.** *If $\Gamma, x : A \vdash M : B$ and $\Gamma \vdash N : A$, then $\Gamma \vdash M[N/x] : B$.*

*Proof.* By induction on the structure of $M$.

- If $M = x$, then $M[N/x] = N$ and we must have that $A = B$. Then, by assumption, we have $\Gamma \vdash N : B = \Gamma \vdash M[N/x] : B$. If $M = y \neq x$, then $M[N/x] = y$ and we get $\Gamma \vdash M : B$ by assumption (note that we can remove $x : A$ from the context, since it was not used in the proof of $\Gamma \vdash y : B$).
- If $M = \lambda y.M_1$ ($y \neq x$ since variable names are chosen maximally fresh), then $(\lambda y.M_1)[N/x] = \lambda y.M_1[N/x]$ and $B = C \rightarrow D$. By the $\rightarrow_I$-rule we get $\Gamma, x : A, y : C \vdash M_1 : D$ and by the induction hypothesis we have $\Gamma, y : C \vdash M_1[N/x] : D$. Then, by the $\rightarrow_I$ rule, we get $\Gamma \vdash \lambda y.M_1[N/x] : C \rightarrow D$, as required.
- If $M = M_1 M_2$, then $(M_1 M_2)[N/x] = (M_1[N/x])(M_2[N/x])$ and the result follows immediately from the induction hypothesis.

□

**Lemma 3.** *The reductions in $\lambda$ satisfy subject reduction. That is: if $\Gamma \vdash M : B$ and $M \to M'$, then also $\Gamma \vdash M' : B$.*

*Proof.* If $M = (\lambda x.M_1)N$, then $M \to_\beta M_1[N/x]$ and we have:

$$\dfrac{\dfrac{\Gamma, x : A \vdash M_1 : B}{\Gamma \vdash \lambda x.M_1 : A \to B}\ {\to_I} \qquad \Gamma \vdash N : A}{\Gamma \vdash (\lambda x.M_1)N : B}\ {\to_E}$$

By Lemma 2 we get $\Gamma \vdash M_1[N/x] : B$. □

A term may contain multiple redexes which can all be contracted in an arbitrary order. The choice to contract a particular redex gives rise to a *reduction path*. In the case of Example 2 we have two reduction paths and both paths yield the same result. This is not a coincidence, since $\lambda_\to$ is *confluent* (see Lemma 4).

**Lemma 4.** *Reduction in $\lambda_\to$ is confluent. That is: if $M \to^* M_1$ and $M \to^* M_2$, then there exists an $N \in \Lambda$ such that $M_1 \to^* N$ and $M_2 \to^* N$.*

*Proof.* We obtain confluence for $\lambda_\to$ by confluence of the untyped $\lambda$-calculus (see e.g. [Tak89]) and Lemma 3. □

In Definition 3 we defined a non-deterministic *reduction system* called *call-by-name*. The idea behind the call-by-name reduction is that arguments to a function are never evaluated before copying them into the body of the function. The obvious drawback of this strategy is that an argument must be evaluated multiple times if it occurs more than once in the body of a function.

Another reduction system is called *call-by-value*. The idea behind call-by-value reduction is that arguments to a function are evaluated to *values* before copying them into the body of the function. The drawback of such system is that arguments are evaluated independent of whether they are used or not. Even if a function throws away its argument, the argument is evaluated.

**Definition 5.** The set $V$ of $\lambda_\to$ values is defined by the following grammar:

$$V ::= x \ \mid \ \lambda x.N$$

Now we must also adapt our $\beta$-reduction rule, since it allows arbitrary arguments (i.e. not necessarily values) to be copied into the body of the function.

**Definition 6.** The call-by-value reduction system for $\lambda_\to$ is defined as the compatible closure of the following reduction:

$$(\lambda x.M)V \to_{\beta_v} M[V/x]$$

The difference with the call-by-name system is that arguments must now be reduced to values. Consider, for example, the term $M = (\lambda x.x)((\lambda y.y)(\lambda z.z))$, which may be reduced in two ways using the call-by-name reduction system (see Example 2). Using the call-by-value system we can only get the following reduction path: $M \to_{\beta_v} (\lambda x.x)(\lambda z.z) \to_{\beta_v} \lambda z.z$.

There are other reduction systems like, for instance, call-by-need (see [MOW98]). We will, however, limit our discussion to the call-by-value and call-by-name systems.

### 2.1.1 Abstract machine

We have already mentioned, in the introduction, that lambda calculi are often used as a foundation for functional programming languages. A term in a calculus expresses a program and, like for any other programming language, we want a machine that can evaluate this program.

While we will encounter machines that are non-deterministic later on in this thesis, we usually want a machine to be deterministic. To that end one must first define an *reduction strategy*. The difference between a *reduction system* and a *reduction strategy* is that the former may be non-deterministic, while the latter is deterministic.

In the case of the $\lambda_{\to}$ we define the (call-by-name) weak head reduction strategy, which is implemented by a machine called $\mathcal{K}$. This machine was first defined by Krivine [Kri07] and, hence, is also referred to as the Krivine-machine. For brevity, we will only discuss this call-by-name machine. It is possible, however, to define a machine that implements a call-by-value reduction strategy (see e.g. [Lan64, Plo75]). Also see [DF07] for a nice discussion on Krivine machines.

**Definition 7.** The (call-by-name) weak head reduction strategy for the $\lambda_{\to}$-calculus is defined by the following rules:

- $(\lambda x.M)N \to_{wh} M[N/x]$
- $M \to_{wh}^{*} M$
- $\dfrac{M \to_{wh} N}{MO \to_{wh} NO}$
- $\dfrac{M \to_{wh} N \quad N \to_{wh}^{*} O}{M \to_{wh}^{*} O}$

Indeed, the weak head reduction strategy never reduces under a $\lambda$-abstraction. That is: if $t = \lambda x.(\lambda y.y)x$, then $t \not\to_{wh}$ even though $(\lambda y.y)x$ is a redex. Accordingly, if the $\mathcal{K}$ machine is loaded with the term $t$, it should halt immediately.

The $\mathcal{K}$ machine is defined in terms of reductions on states of the following form: $\langle M,\ E,\ S\rangle$. Here, $E$ is an *environment* mapping variables to *closures* $\langle N, E'\rangle$, where $N$ is a term and $E'$ is another environment. We write $E[x \mapsto \langle N, E'\rangle]$ for the environment mapping $x$ to the closure $\langle N, E'\rangle$ and $E(x) = \langle N, E'\rangle$. On the *stack* $S$ we can store closures. We write $\langle N, E\rangle :: S$ for the stack obtained by pushing the closure $\langle N, E\rangle$ on of top the stack $S$.

**Definition 8.** The $\mathcal{K}$ machine is defined by the following reductions:

1. $\langle x,\ E,\ S\rangle \to \langle N,\ E',\ S\rangle$, if $E(x) = \langle N, E'\rangle$
2. $\langle \lambda x.M,\ E,\ \langle N, E'\rangle :: S\rangle \to \langle M,\ E[x \mapsto \langle N, E'\rangle],\ S\rangle$
3. $\langle MN,\ E,\ S\rangle \to \langle M,\ E,\ \langle N, E\rangle :: S\rangle$

Evaluation of a term always starts with the empty environment "-" and the empty stack "-".

**Example 3.** Consider the following example, in which we show how the $\mathcal{K}$ machine reduces the term $KI\Omega$, where $K = \lambda x.\lambda y.x$, $I = \lambda z.z$ and $\Omega = (\lambda a.aa)(\lambda b.bb)$:

$$\begin{aligned}\langle KI\Omega,\ -,\ -\rangle &\to \langle KI,\ -,\ \langle \Omega, -\rangle\rangle \\ &\to \langle K,\ -,\ \langle I, -\rangle :: \langle \Omega, -\rangle\rangle \\ &\to \langle \lambda y.x,\ [x \mapsto \langle I, -\rangle],\ \langle \Omega, -\rangle\rangle \\ &\to \langle x,\ [x \mapsto \langle I, -\rangle][y \mapsto \langle \Omega, -\rangle],\ -\rangle \\ &\to \langle I,\ -,\ -\rangle\end{aligned}$$

*end of example*

**Lemma 5.** *The $\mathcal{K}$ machine is correct for the weak head reduction strategy in $\lambda_\to$. That is: if $\mathcal{M}$ is a machine state and $\mathcal{M} \to \mathcal{M}'$, then $[\![\mathcal{M}]\!] \to_{wh}^* [\![\mathcal{M}']\!]$. Where $[\![-]\!]$ maps machine states to $\lambda$ terms.*

*Proof.* This is proven in [Kri07]. □

Usually we would also like to prove that the machine is complete for its reduction strategy. That is: if $M$ is a $\lambda$ term and $M \to_{wh} M'$, then $\{\!\{M\}\!\} \to^* \{\!\{M\}\!\}'$, where $\{\!\{-\}\!\}$ maps $\lambda$ terms to machine states. This, however, is not so easy to prove. The difficulty is that, for an arbitrary $\lambda$ term, we simply do not know which closures should be put in the environment. We will discuss a solution to this problem in the next section.

## 2.2 The $\lambda\mu$-calculus

In this section we recall the $\lambda\mu$-calculus, which was first introduced by Parigot in [Par92]. We use the same presentation as Curien and Herbelin in [CH00], as this makes it easier to express similarities between $\lambda\mu$ and $\bar{\lambda}\mu$ later on. The syntax of $\lambda\mu$ is divided into the following two categories.

**Definition 9.** The set $\Lambda\mu$ of $\lambda\mu$ terms is defined by the following grammars:

$$\begin{array}{lrcl} \text{Terms} & M, N & ::= & x \mid \lambda x.M \mid \mu\alpha.c \mid MN \\ \text{Commands} & c & ::= & [\beta]M \end{array}$$

The syntax uses two sets of variables: we let $x, y, \ldots$ range over the set of term-variables $\mathcal{X}$ and let $\alpha, \beta, \ldots$ range over the set of $\mu$-variables $\mathcal{A}$. The construct $\lambda x.M$ binds $x$ in $M$ and the construct $\mu\alpha.c$ binds $\alpha$ in $c$.

The typing system uses two kinds of judgements, one for each syntactic category:

$$c : (\Gamma \vdash \Delta) \qquad \Gamma \vdash M : A \mid \Delta$$

Here, $\Gamma$ is a context (see Section 2.1) containing term-variables and $\Delta$ is a context containing $\mu$-variables. The typing rules are included in Figure 2.2. Here, the "|" in the judgements serves to single out a specific (active) formula.

$$\frac{}{\Gamma, x : A \vdash x : A \mid \Delta}\ \text{AX}$$

$$\frac{\Gamma, x : A \vdash M : B \mid \Delta}{\Gamma \vdash \lambda x.M : A \to B \mid \Delta}\ \to_I \qquad \frac{\Gamma \vdash M : A \to B \mid \Delta \quad \Gamma \vdash N : A \mid \Delta}{\Gamma \vdash MN : B \mid \Delta}\ \to_E$$

$$\frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha.c : A \mid \Delta}\ \mu \qquad \frac{\Gamma \vdash M : A \mid \alpha : A, \Delta}{[\alpha]M : (\Gamma \vdash \alpha : A, \Delta)}\ \text{CM}$$

Figure 2.2: Typing rules for $\lambda\mu$

**Example 4.** Consider the following derivation, which shows that the term $\lambda x.\mu\alpha.[\alpha]x(\lambda y.\mu\beta.[\alpha]y)$ can be typed with Peirce's law.

$$\dfrac{\dfrac{}{x : (A \to B) \to A \vdash x : (A \to B) \to A \mid \alpha : A}\ \text{AX} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{x : (A \to B) \to A, y : A \vdash y : A \mid \beta : B, \alpha : A}\ \text{AX}}{[\alpha]y : (x : (A \to B) \to A, y : A \vdash \beta : B, \alpha : A)}\ \text{CM}}{x : (A \to B) \to A, y : A \vdash \mu\beta.[\alpha]y : B \mid \alpha : A}\ \mu}{x : (A \to B) \to A \vdash (\lambda y.\mu\beta.[\alpha]y) : A \to B \mid \alpha : A}\ \to_I}{\dfrac{\dfrac{\dfrac{x : (A \to B) \to A \vdash x(\lambda y.\mu\beta.[\alpha]y) : A \mid \alpha : A}{([\alpha]x(\lambda y.\mu\beta.[\alpha]y)) : (x : (A \to B) \to A \vdash \alpha : A)}\ \text{CM}}{x : (A \to B) \to A \vdash \mu\alpha.[\alpha]x(\lambda y.\mu\beta.[\alpha]y) : A \mid}\ \mu}{\vdash \lambda x.\mu\alpha.[\alpha]x(\lambda y.\mu\beta.[\alpha]y) : ((A \to B) \to A) \to A \mid}\ \to_I}\ \to_E$$

*end of example*

From a logical perspective, if we look at the typing rules in a bottom-up fashion, the $\mu$-operator allows us to "store" the formula we are currently proving into the context $\Delta$. That is: if we have to prove $\Gamma \vdash \mu\alpha.c : A$, we may store the formula $A$ under the name $\alpha$ in $\Delta$ and continue with proving $c : (\Gamma \vdash \Delta, \alpha : A)$. Now, if we encounter a command and must prove e.g. $([\beta]M) : (\Gamma \vdash \Delta)$ we can "restore" the formula that was previously saved under the name $\beta$ in $\Delta$. I.e. a command allows us to "activate" a formula that we made "passive" earlier on.

Traditionally, $\lambda\mu$ is equipped with a call-by-name reduction system. While there also exists a call-by-value variant of the $\lambda\mu$-calculus (see e.g. [Nak03]), we will limit our discussion to the call-by-name case.

**Definition 10.** Reduction in $\lambda\mu$ is defined as the congruent closure of the following rules:

$$\begin{array}{ll} (\beta) & (\lambda x.M)N \to M[N/x] \\ (\mu) & (\mu\alpha.c)N \to \mu\alpha.c[[\alpha]MN/[\alpha]M] \\ (\mu R) & [\alpha](\mu\beta.c) \to c[\alpha/\beta] \end{array}$$

From a computational perspective, a $\mu$-abstraction allows us to pass its arguments through to one of its sub-terms. This functionality is reflected in the $\mu$-reduction rule. If we encounter a term like $(\mu\alpha.c)N_1 \dots N_k$, then we may pass the arguments $N_1 \dots N_k$ to the sub-term named $\alpha$, i.e. the command $[\alpha]M$ where $M$ is an arbitrary term, so that we eventually obtain the command $[\alpha]MN_1 \dots N_k$. In Section 2.2.1 we will see that the $\mu$-operator can also be used to create an exception handling mechanism.

**Example 5.** Consider, for example, the following reduction sequence:

$$\begin{aligned}(\mu\alpha.[\alpha](\mu\beta.[\alpha]x))yz &\to_\mu (\mu\alpha.[\alpha](\mu\beta.[\alpha]xy))z \\ &\to_\mu \mu\alpha.[\alpha](\mu\beta.[\alpha]xyz) \\ &\to_{\mu R} \mu\alpha.[\alpha]xyz\end{aligned}$$

*end of example*

On top of the reductions from Definition 10, we may add the following $\eta$-like reduction:

$$\mu\alpha.[\alpha]M \to_\eta M \quad \text{if } \alpha \notin FV(M)$$

The $\lambda\mu$-calculus satisfies all the major properties:

**Lemma 6.** *The reductions in $\lambda\mu$ satisfy subject reduction. That is: if $\Gamma \vdash M : A \mid \Delta$ and $M \to M'$, then also $\Gamma \vdash M' : A \mid \Delta$.*

*Proof.* See e.g. [Kre10]. □

**Lemma 7.** *The reductions in $\lambda\mu$ are strongly normalizing. That is: the process of reduction always terminates.*

*Proof.* See [Par97]. □

**Lemma 8.** *The reductions in $\lambda\mu$ are confluent. That is: if $M \in \Lambda\mu$ and $M \to^* M_1$ and $M \to^* M_2$, then there exists a term $M'$ such that $M_1 \to^* M'$ and $M_2 \to M'$.*

*Proof.* See [Par92] (also see [Kre10]). □

### 2.2.1 Exception handling

In $\lambda\mu$ we can use the $\mu$-abstraction to create an exception handling mechanism. We can define the control operators `catch` and `throw` as follows.

**Definition 11.** The control operators `catch` and `throw` are defined as follows:

$$\begin{aligned}&\texttt{catch}_\alpha\ M = \mu\alpha.[\alpha]M \\ &\texttt{throw}_\beta\ M = \mu\gamma.[\beta]M \quad \text{if } \gamma \notin FV(M)\end{aligned}$$

**Lemma 9.** *We have the following reductions for `catch` and `throw` [Kre10]:*

1. $(\texttt{throw}_\alpha\ M)N_1 \dots N_k \to \texttt{throw}_\alpha\ M$
2. $\texttt{catch}_\alpha\ (\texttt{throw}_\alpha\ M) \to \texttt{catch}_\alpha\ M$
3. $\texttt{catch}_\alpha\ (\texttt{throw}_\beta\ M) \to \texttt{throw}_\beta\ M$, *provided that* $\alpha \notin FV(M)$
4. $\texttt{throw}_\alpha\ (\texttt{throw}_\beta\ M) \to \texttt{throw}_\beta\ M$
5. $\texttt{catch}_\alpha\ M \to M$, *provided that* $\alpha \notin FV(M)$

*Proof.*

1. For reduction (1) we have:

$$\begin{aligned}(\texttt{throw}_\alpha\ M)N_1 \dots N_k &= (\mu\gamma.[\alpha]M)N_1 \dots N_k \ \text{ with } \gamma \notin FV(M)\\ &\to_\mu (\mu\gamma.[\alpha]M)N_2 \dots N_k\\ &\to_\mu \dots \to_\mu (\mu\gamma.[\alpha]M)N_k\\ &\to_\mu (\mu\gamma.[\alpha]M) = \texttt{throw}_\alpha\ M\end{aligned}$$

2. For reduction (2) we have:

$$\begin{aligned}\texttt{catch}_\alpha\ (\texttt{throw}_\alpha\ M) &= \mu\alpha.[\alpha](\mu\gamma.[\alpha]M) \ \text{ with } \gamma \notin FV(M)\\ &\to_{\mu R} \mu\alpha.[\alpha]M = \texttt{catch}_\alpha\ M\end{aligned}$$

3. For reduction (3) we have:

$$\begin{aligned}\texttt{catch}_\alpha\ (\texttt{throw}_\beta\ M) &= \mu\alpha.[\alpha](\mu\gamma.[\beta]M) \ \text{ with } \gamma \notin FV(M)\\ &\to_{\mu R} \mu\alpha.[\beta]M = \texttt{throw}_\beta\ M \ \text{ if } \alpha \notin FV(M)\end{aligned}$$

4. For reduction (4) we have:

$$\begin{aligned}\texttt{throw}_\alpha\ (\texttt{throw}_\beta\ M) &= \mu\gamma.[\alpha](\mu\delta.[\beta]M) \ \text{ with } \gamma, \delta \notin FV(M)\\ &\to_{\mu R} \mu\gamma.[\beta]M = \texttt{throw}_\beta\ M\end{aligned}$$

5. For reduction (5) we have:

$$\begin{aligned}\texttt{catch}_\alpha\ M &= \mu\alpha.[\alpha]M\\ &\to_\eta M \ \text{ if } \alpha \notin FV(M)\end{aligned}$$

□

The intuition behind this exception handling mechanism is that we want a `throw` to propagate upwards. That is: if some computation results in an exception $\texttt{throw}_\alpha\ M$, we want to keep throwing this exception upwards until we find a suitable `catch` (i.e. $\texttt{catch}_\alpha$ ). If, eventually, a suitable `catch` is found (reduction (2)), we do not want to remove it, we only want to remove the exception, since the body of the exception itself may still contain exceptions that we might want to handle. Reductions (1), (3) and (4) are responsible for propagating an exception upwards, reduction (2) is responsible for "handling" an exception and reduction (5) is responsible for removing the top-level `catch` , if there is one.

**Example 6.** Consider reducing the following term using these new reduction rules:

$$\begin{aligned}\texttt{catch}_\alpha\ ((\lambda x.\texttt{throw}_\alpha\ x)(\lambda y.y)) &\to \texttt{catch}_\alpha\ (\texttt{throw}_\alpha\ (\lambda y.y))\\ &\to \texttt{catch}_\alpha\ (\lambda y.y)\\ &\to \lambda y.y\end{aligned}$$

*end of example*

### 2.2.2 Abstract machine

Just like for the $\lambda$-calculus, we can define an abstract machine that evaluates $\lambda\mu$-terms. For the call-by-name $\lambda\mu$-calculus it is possible to define an extension of the Krivine machine $\mathcal{K}$ from Section 2.1.1. Accordingly, the machine, which is defined e.g. in [DG98], is called $\mu\mathcal{K}$.

**Definition 12.** The weak head reduction strategy for $\lambda\mu$ is defined by the following rules (also see [DG98]):

- $(\lambda x.M)N \to_{wh} M[N/x]$
- $(\mu\alpha.c)N \to_{wh} \mu\alpha.c[[\alpha]MN/[\alpha]M]$
- $\mu\alpha.\mu\beta.c \to_{wh} \mu\alpha.c[\alpha/\beta]$
- $\mu\alpha.[\beta](\mu\gamma.c) \to_{wh} \mu\alpha.c[\beta/\gamma]$
- $\mu\alpha.[\alpha]M \to_{wh} M$, if $\alpha \notin FV(M)$
- $M \to^*_{wh} M$
- $\dfrac{M \to_{wh} N}{MO \to_{wh} NO}$
- $\dfrac{M \to_{wh} N \quad N \to^*_{wh} O}{M \to^*_{wh} O}$
- $\dfrac{M \to_{wh} N}{\mu\alpha.M \to_{wh} \mu\alpha.N}$
- $\dfrac{M \to_{wh} N}{[\alpha]M \to_{wh} [\alpha]N}$

**Definition 13.** The $\mu\mathcal{K}$-machine [DG98] is defined by the following reductions:

1. $\langle x,\ E,\ S\rangle \to \langle N,\ E',\ S\rangle$, if $E(x) = \langle N, E'\rangle$
2. $\langle \lambda x.M,\ E,\ \langle N, E'\rangle :: S\rangle \to \langle M,\ E[x \mapsto \langle N, E'\rangle],\ S\rangle$
3. $\langle MN,\ E,\ S\rangle \to \langle M,\ E,\ \langle N, E\rangle :: S\rangle$
4. $\langle \mu\alpha.c,\ E,\ S\rangle \to \langle c,\ E[\alpha \mapsto S],\ -\rangle$
5. $\langle [\alpha]M,\ E,\ -\rangle \to \langle M,\ E,\ S\rangle$, if $E(\alpha) = S$

**Example 7.** Consider the following machine reduction:

$$
\begin{aligned}
&\langle(\mu\alpha.[\alpha](\mu\beta.[\alpha](\lambda x.x)))(\lambda y.y),\ -,\ -\rangle \\
&\to \langle \mu\alpha.[\alpha](\mu\beta.[\alpha](\lambda x.x)),\ -,\ \langle \lambda y.y, -\rangle\rangle \\
&\to \langle [\alpha](\mu\beta.[\alpha](\lambda x.x)),\ [\alpha \mapsto \langle \lambda y.y, -\rangle],\ -\rangle \\
&\to \langle \mu\beta.[\alpha](\lambda x.x),\ [\alpha \mapsto \langle \lambda y.y, -\rangle],\ \langle \lambda y.y, -\rangle\rangle \\
&\to \langle [\alpha](\lambda x.x),\ [\alpha \mapsto \langle \lambda y.y, -\rangle][\beta \mapsto \langle \lambda y.y, -\rangle],\ -\rangle \\
&\to \langle \lambda x.x,\ [\alpha \mapsto \langle \lambda y.y, -\rangle][\beta \mapsto \langle \lambda y.y, -\rangle],\ \langle \lambda y.y, -\rangle\rangle \\
&\to \langle x,\ [\alpha \mapsto \langle \lambda y.y, -\rangle][\beta \mapsto \langle \lambda y.y, -\rangle][x \mapsto \langle \lambda y.y, -\rangle],\ -\rangle \\
&\to \langle \lambda y.y,\ -,\ -\rangle
\end{aligned}
$$

*end of example*

**Lemma 10.** *The machine is correct for the weak head reduction strategy in $\lambda\mu$. That is: if $\mathcal{M}$ is a machine state and $\mathcal{M} \to \mathcal{M}'$, then $[\![\mathcal{M}]\!] \to_{wh}^* [\![\mathcal{M}']\!]$. Where $[\![-]\!]$ maps machine states to $\lambda$ terms.*

*Proof.* This is proven in [DG98]. □

In [DG98], De Groote also proves that the $\mu\mathcal{K}$-machine is complete for the weak head reduction strategy in $\lambda\mu$. This result is not obtained by defining a map, $\{\!\{\}\!\}$ say, from $\lambda\mu$-terms to $\mu\mathcal{K}$ machine states. Like we mentioned in Section 2.1.1, the difficulty in defining such map is that, for arbitrary terms, we don't know what to put into the machines environment. A simplification could be to concentrate on closed, well-typed terms only, because then we can translate any such term $t$ to the machine state $\langle t,\ -,\ -\rangle$. However, this will not work either. Consider, for example, the closed, well-typed term $(\lambda y.\lambda x.x)M \to_{wh} (\lambda x.x)$. We would like to have that $\langle(\lambda y.\lambda x.x)M,\ -,\ -\rangle \to^* \langle\lambda x.x,\ -,\ -\rangle$. However, this is not the case, since $\langle(\lambda y.\lambda x.x)M,\ -,\ -\rangle \to \langle\lambda y.\lambda x.x,\ -,\ \langle M, -\rangle\rangle \to \langle\lambda x.x,\ [y \mapsto \langle M, -\rangle],\ -\rangle \not\to$. De Groote's solution is to prove the following lemma.

**Lemma 11.** *The $\mu\mathcal{K}$-machine is complete for the weak head reduction strategy in $\lambda\mu$. That is: for any closed, well-typed term $M$ of type $\iota$, there exists an environment $E$ such that $\langle M,\ -,\ -\rangle \to^* \langle\star,\ E,\ -\rangle$. Where $\star$ is a constant of type $\iota$. Moreover, $M \to_{wh}^* [\![\langle\star,\ E,\ -\rangle]\!]$. I.e. the following diagram commutes:*

$$\begin{array}{ccc} M & \xrightarrow{\{\!\{-\}\!\}} & \langle M,\ -,\ -\rangle \\ \Big\downarrow{\scriptstyle wh^*} & & \Big\downarrow{\scriptstyle *} \\ [\![\langle\star,\ E,\ -\rangle]\!] & \xleftarrow[[\![-]\!]]{} & \langle\star,\ E,\ -\rangle \end{array}$$

*Proof.* This is proven in [DG98]. □

# Chapter 3

# The $\bar{\lambda}\mu$-calculus

In this chapter we investigate some properties of the $\bar{\lambda}\mu$-calculus, which was first introduced by Herbelin in [Her95]. The calculus can be seen as a sequent-calculus version of Parigot's call-by-name $\lambda\mu$ and, hence, has a lot in common with $\lambda\mu$. In fact, the calculi turn out to be isomorphic [CH00]. In Section 3.1 we will define the inverse of the map $\mathcal{N}$ (which is defined in [CH00]) and show that reduction and typing is preserved.

The syntax of $\bar{\lambda}\mu$ is divided into the following three categories.

**Definition 14.** The set $\bar{\Lambda}\mu$ of $\bar{\lambda}\mu$ terms is defined by the following grammars:

$$\begin{array}{lllll} \text{Commands} & c & ::= & \langle v \parallel e \rangle & \\ \text{Terms} & v & ::= & x \mid \lambda x.v \mid \mu\alpha.c & \\ \text{Contexts} & e & ::= & \beta \mid v \cdot e & \end{array}$$

We let $t$ range over arbitrary $\bar{\lambda}\mu$ terms i.e. $t \in \bar{\Lambda}\mu$. The syntax uses two sets of variables: we let $x, y, \ldots$ range over the set of term-variables $\mathcal{X}$ and let $\alpha, \beta, \ldots$ range over the set of context-variables $\mathcal{A}$. The construct $\lambda x.v$ binds $x$ in $v$ and the construct $\mu\alpha.c$ binds $\alpha$ in $c$. As usual, we will associate terms that only differ in the names of their bound variables. That is: we work on $\alpha$-equivalence classes ($\bar{\Lambda}\mu/\equiv$) of terms. Moreover, we adopt the convention that the names of bound variables are chosen maximally fresh and different from the names of free variables.

**Definition 15.** The set of free variables $FV(t)$ of a $\bar{\lambda}\mu$ term $t$ can be determined by the following function:

$$\begin{aligned} FV(\nu) &= \{\nu\} \quad \text{, where } \nu \text{ is either in } \mathcal{X} \text{ or in } \mathcal{A} \\ FV(\lambda x.v) &= FV(v) - \{x\} \\ FV(\mu\alpha.c) &= FV(c) - \{\alpha\} \\ FV(v \cdot e) &= FV(v) \cup FV(e) \\ FV(\langle v \parallel e \rangle) &= FV(v) \cup FV(e) \end{aligned}$$

The typing system uses three kinds of judgements, one for each syntactic category:

$$c : (\Gamma \vdash \Delta) \qquad \Gamma \vdash v : A \mid \Delta \qquad \Gamma \mid e : A \vdash \Delta$$

The typing rules for $\bar{\lambda}\mu$ are included in Figure 3. Here, again, the "|" serves to single out a specific (active) formula. For details about the underlying logic we refer to [CH00, Her95].

$$\frac{}{\Gamma \mid \alpha : A \vdash \alpha : A, \Delta}\ \text{AX}_l \qquad \frac{}{\Gamma, x : A \vdash x : A \mid \Delta}\ \text{AX}_r$$

$$\frac{\Gamma \vdash v : A \mid \Delta \quad \Gamma \mid e : B \vdash \Delta}{\Gamma \mid (v \cdot e) : A \to B \vdash \Delta}\ \to_l \qquad \frac{\Gamma, x : A \vdash v : B \mid \Delta}{\Gamma \vdash \lambda x.v : A \to B \mid \Delta}\ \to_r$$

$$\frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha.c : A \mid \Delta}\ \mu \qquad \frac{\Gamma \vdash v : A \mid \Delta \quad \Gamma \mid e : A \vdash \Delta}{\langle v \parallel e \rangle : (\Gamma \vdash \Delta)}\ \text{CM}$$

Figure 3.1: Typing rules for $\bar{\lambda}\mu$

**Fact 1.** Proofs in $\bar{\lambda}\mu$ will always be closed by either the $\text{AX}_l$ or $\text{AX}_r$ axiom from Figure 3. In these axioms both $\Gamma$ and $\Delta$ can be arbitrarily large. This allows us to show that the following weakening rules are derivable (where $x \notin \Gamma$ and $\alpha \notin \Delta$):

$$\frac{\Gamma \mid e : A \vdash \Delta}{\Gamma, x : B \mid e : A \vdash \Delta}\ \text{WK}_{\Gamma l} \qquad \frac{\Gamma \vdash v : A \mid \Delta}{\Gamma, x : B \vdash v : A \mid \Delta}\ \text{WK}_{\Gamma r}$$

$$\frac{\Gamma \mid e : A \vdash \Delta}{\Gamma \mid e : A \vdash \alpha : B, \Delta}\ \text{WK}_{\Delta l} \qquad \frac{\Gamma \vdash v : A \mid \Delta}{\Gamma \vdash v : A \mid \alpha : B, \Delta}\ \text{WK}_{\Delta r}$$

A context $e$ can be read as a term with a "hole" ([ ]) in it. This hole can be interpreted as the absence of a computation. In that sense, contexts are the dual of terms: while terms express programs or computations, contexts express a blocked computation waiting for the result of a sub-computation. We often write $e[t]$ to denote the term obtained by replacing (filling) the "hole" in $e$ with the term $t$, where free variables in $t$ may become bound in $e[t]$.

A command $\langle v \parallel e \rangle$ can be read as a term $v$ that is put in the hole of the context $e$, it is the equivalent of $e[v]$. If $e = \alpha$, then $\langle v \parallel e \rangle$ is just another notation for the naming construct $[\alpha]v$ from the $\lambda\mu$-calculus. The intuition for the typing of contexts in the CM and $\to_l$ rules from Figure 3 is as follows: context $e$ is of type $B$ if the "hole" in $e$ is of type $B$.

The construction $v \cdot e$ can be interpreted as an applicative context. It is the $\bar{\lambda}\mu$ equivalent of filling the hole in $e$ with the application of a new hole and $v$ i.e. $e[[\,]v]$. We can now encode a function that is applied to a list of arguments in context $e$ as follows:

$$e[(\lambda x_1, \ldots, x_k.N)M_1 \ldots M_k] = \langle \lambda x_1, \ldots, x_k.N \parallel M_1 \cdot \ldots \cdot M_k \cdot e \rangle$$

**Definition 16.** We have the following reductions in $\bar{\lambda}\mu$:

$$\begin{array}{lrcl} (\lambda) & \langle \lambda x.v_1 \parallel v_2 \cdot e \rangle & \to & \langle v_1[v_2/x] \parallel e \rangle \\ (\mu) & \langle \mu\alpha.c \parallel e \rangle & \to & c[e/\alpha] \end{array}$$

Where substitution is capture-free with respect to both kinds of variables.

**Definition 17.** Substitution $t[t'/\nu]$ is defined as follows:

$$\begin{aligned} x[t/\nu] &= t \quad \text{, if } x = \nu \\ y[t/\nu] &= y \quad \text{, if } y \neq \nu \\ (\lambda y.v)[t/\nu] &= \lambda y.v[t/\nu] \\ (\mu\alpha.c)[t/\nu] &= \mu\alpha.c[t/\nu] \\ (v \cdot e)[t/\nu] &= (v[t/\nu]) \cdot (e[t/\nu]) \\ \langle v \parallel e \rangle[t/\nu] &= \langle v[t/\nu] \parallel e[t/\nu] \rangle \end{aligned}$$

**Remark 1.** Note that the $\lambda$-reduction indeed conforms to the explanation of $\langle M \parallel N \cdot e \rangle$ given above.

On top of the reduction rules from Definition 16 we could add an $\eta$-equality similar to the one of $\lambda\mu$:

$$\mu\alpha.\langle v \parallel \alpha \rangle =_\eta v, \text{ if } \alpha \text{ is not free in } v$$

Let us check that the reductions in $\bar{\lambda}\mu$ satisfy subject reduction. First we need to establish that typing is preserved under substitution, as the following lemma shows.

**Lemma 12.** *Typing is preserved under substitution. That is: if*

*(A)* $\Gamma \mid e' : A' \vdash \Delta$,

*(B)* $\Gamma \vdash v : A \mid \alpha : A', \Delta$ *and*

*(C)* $\Gamma \mid e : A \vdash \alpha : A', \Delta$

*then the following statements hold:*

*(1)* $\Gamma \mid e[e'/\alpha] : A \vdash \Delta$

*(2)* $\Gamma \vdash v[e'/\alpha] : A \mid \Delta$

*Proof.* Simultaneously, by induction on the structure of $e, v$.

- (1) If $e = \beta$ with $\beta \neq \alpha$, then $e[e'/\alpha] = \beta[e'/\alpha] = \beta$ and, hence $\Gamma \mid e[e'/\alpha] : A \vdash \Delta$ by (C) (note that we can use the weakening rule $\text{WK}_{\Delta l}$ to remove $\alpha : A'$ from the context).
  If $e = \alpha$, then $A = A'$ and $e[e'/\alpha] = e'$ and, hence, we get $\Gamma \mid e[e'/\alpha] : A \vdash \Delta$ by (A).

  (2) If $v = x$, then $v[e'/\alpha] = x$ and, hence, we have $\Gamma \vdash v[e'/\alpha] : A \mid \Delta$ by (B). As before, we can remove $\alpha : A'$ from the context using $\text{WK}_{\Delta r}$.

- (1) If $e = v' \cdot e''$, then we must have that $A = B \to C$ for some $B$ and $C$. By (C) and the $\to_l$ tying rule we get $\Gamma \vdash v' : B \mid \alpha : A', \Delta$ and $\Gamma \mid e'' : C \vdash \alpha : A', \Delta$. By applying the induction hypothesis to these judgements we get $\Gamma \vdash v'[e'/\alpha] : B \mid \Delta$ and $\Gamma \mid e''[e'/\alpha] : C \vdash \Delta$.

Applying the $\to_l$ rule gives us the desired result ($e[e'/\alpha] = v'[e'/\alpha] \cdot e''[e'/\alpha]$):

$$\frac{\Gamma \vdash v'[e'/\alpha] : B \mid \Delta \quad \Gamma \mid e''[e'/\alpha] : C \vdash \Delta}{\Gamma \mid v'[e'/\alpha] \cdot e''[e'/\alpha] : B \to C \vdash \Delta} \to_l$$

(2) If $v = \lambda x.v'$, then $A = B \to C$ for some $B$ and $C$. By (B) and the $\to_r$ typing rule we get $\Gamma, x : B \vdash v' : C \mid \alpha : A', \Delta$. By applying the induction hypothesis we get $\Gamma, x : B \vdash v'[e'/\alpha] : C \mid \Delta$. Then, by the $\to_r$ rule, we get the following derivation, as desired ($v[e'/\alpha] = \lambda x.v'[e'/\alpha]$):

$$\frac{\Gamma, x : B \vdash v'[e'/\alpha] : C \mid \Delta}{\Gamma \vdash \lambda x.v'[e'/\alpha] : B \to C \mid \Delta} \to_r$$

- (2) If $v = \mu\beta.\langle v' \parallel e''\rangle$, then we get $\Gamma \vdash v' : B \mid \beta : A, \alpha : A', \Delta$ and $\Gamma \mid e'' : B \vdash \beta : A, \alpha : A', \Delta$ by applying first the $\mu$ and then the CM rule for some $B$. By the induction hypothesis we get $\Gamma \vdash v'[e'/\alpha] : B \mid \beta : A, \Delta$ and $\Gamma \mid e''[e'/\alpha] : B \vdash \beta : A, \Delta$. We now arrive at the following derivation, as desired ($v[e'/\alpha] = (\mu\beta.\langle v' \parallel e''\rangle)[e'/\alpha] = \mu\beta.\langle v'[e'/\alpha] \parallel e''[e'/\alpha]\rangle$):

$$\frac{\dfrac{\Gamma \vdash v'[e'/\alpha] : B \mid \beta : A, \Delta \quad \Gamma \mid e''[e'/\alpha] : B \vdash \beta : A, \Delta}{\langle v'[e'/\alpha] \parallel e''[e'/\alpha]\rangle : (\Gamma \vdash \beta : A, \Delta)}\ \text{CM}}{\Gamma \vdash \mu\beta.\langle v'[e'/\alpha] \parallel e''[e'/\alpha]\rangle : A \mid \Delta}\ \mu$$

□

**Corollary 1.** *From Lemma 12 it easily follows that if* $\Gamma \mid e' : A' \vdash \Delta$, $\Gamma \vdash v : A \mid \alpha : A', \Delta$ *and* $\Gamma \mid e : A \vdash \alpha : A', \Delta$, *then* $\langle v \parallel e\rangle[e'/\alpha] : (\Gamma \vdash \Delta)$ *holds, since* $\langle v \parallel e\rangle[e'/\alpha] = \langle v[e'/\alpha] \parallel e[e'/\alpha]\rangle$ *and:*

$$\frac{\dfrac{\textit{Lemma 12}}{\Gamma \vdash v[e'/\alpha] : A \mid \Delta} \quad \dfrac{\textit{Lemma 12}}{\Gamma \mid e[e'/\alpha] : A \vdash \Delta}}{\langle v[e'/\alpha] \parallel e[e'/\alpha]\rangle : (\Gamma \vdash \Delta)}\ \text{CM}$$

**Remark 2.** The results from Lemma 12 and Corollary 1 also hold for substitution with terms for term variables i.e. for $v[v'/x]$ and $e[v'/x]$.

**Lemma 13.** *The reductions in* $\bar{\lambda}\mu$ *satisfy subject reduction. That is: if* $\langle v \parallel e\rangle : (\Gamma \vdash \Delta)$ *and* $\langle v \parallel e\rangle \to \langle v' \parallel e'\rangle$, *then* $\langle v' \parallel e'\rangle : (\Gamma \vdash \Delta)$.

*Proof.*

- Suppose $\langle v \parallel e\rangle = \langle \lambda x.v_1 \parallel v_2 \cdot e\rangle$, then $\langle v \parallel e\rangle \to_\lambda \langle v_1[v_2/x] \parallel e\rangle$. We have:
- Suppose $\langle v \parallel e\rangle = \langle \mu\beta.c \parallel e\rangle$, then $\langle v \parallel e\rangle \to_\mu c[e/\beta]$. We have:

$$\frac{\dfrac{c : (\Gamma \vdash \beta : A, \Delta)}{\Gamma \vdash \mu\beta.c : A \mid \Delta}\ \mu \quad \Gamma \mid e : A \vdash \Delta}{\langle \mu\beta.c \parallel e\rangle : (\Gamma \vdash \Delta)}\ \text{CM}$$

Using Corollary 1 we get $c[e/\beta] : (\Gamma \vdash \Delta)$ as desired.

□

$$\dfrac{\dfrac{\Gamma, x : A \vdash v_1 : B \mid \Delta}{\Gamma \vdash \lambda x.v_1 : A \to B \mid \Delta} \to_r \quad \dfrac{\Gamma \vdash v_2 : A \mid \Delta \qquad \Gamma \mid e : B \vdash \Delta}{\Gamma \mid v_2 \cdot e : A \to B \vdash \Delta} \to_l}{\langle \lambda x.v_1 \parallel v_2 \cdot e \rangle : (\Gamma \vdash \Delta)} \text{ CM}$$

$$\longrightarrow_\lambda$$

$$\dfrac{\dfrac{\text{Remark 2}}{\Gamma \vdash v_1[v_2/x] : A \mid \Delta} \qquad \Gamma \mid e : B \vdash \Delta}{\langle v_1[v_2/x] \parallel e \rangle : (\Gamma \vdash \Delta)} \text{ CM}$$

**Lemma 14.** *The reductions in $\bar{\lambda}\mu$ are strongly normalizing. That is: the process of reduction always terminates.*

*Proof.* See [Her95]. □

**Corollary 2.** *The reductions in $\bar{\lambda}\mu$ are trivially locally confluent (there are no critical pairs). Then, using Newman's lemma and Lemma 14, we obtain confluence for well-typed terms in $\bar{\lambda}\mu$.*

So how can we see that the reductions from Definition 16 indeed define a call-by-name strategy? The intuition is that, in a call-by-name calculus, we do not have to start by first reducing arguments (to a function) to values before evaluating the function. Instead we copy arguments directly into the body of the function. With that in mind, let us take a look at the $\to_\lambda$ reduction from Definition 16. Remember that $v \cdot e$ can be interpreted as the context $e[[\,]v]$ and that $\langle v \parallel e \rangle$ can be read as $e[v]$.

Following the $\to_\lambda$ rule we see that $\langle \lambda x.v_1 \parallel v_2 \cdot e \rangle$, which is equivalent to $e[(\lambda x.v_1)v_2]$, reduces to $\langle v_1[v_2/x] \parallel e \rangle$, which is equivalent to $e[v_1[v_2/x]]$. So, indeed, we have the potential to move $v_2$ into the body of $\lambda x.v_1$ without first reducing it to a value.

We also have the option to reduce arguments to values first, as can be seen in Example 8. In $\bar{\lambda}\mu$ we are, however, never forced to do that, as is the case with a call-by-value strategy.

Let us have a look at this from a more formal perspective by defining a map $\delta : \Lambda \to \bar{\Lambda}\mu$. We then show that this map preserves reduction, which shows us that $\bar{\lambda}\mu$ is a true call-by-name calculus. We also show that $\delta$ preserves types.

**Definition 18.**

$$\begin{aligned} \delta(x) &= \mu\alpha.\langle x \parallel \alpha \rangle, \text{ with } \alpha \text{ fresh} \\ \delta(\lambda x.N) &= \mu\alpha.\langle \lambda x.\delta(N) \parallel \alpha \rangle, \text{ with } \alpha \text{ fresh} \\ \delta(MN) &= \mu\alpha.\langle \delta(M) \parallel \delta(N) \cdot \alpha \rangle, \text{ with } \alpha \text{ fresh} \end{aligned}$$

**Fact 2.** It can be easily seen from the definition above that $\delta(M) = \mu\alpha.c$, for all $M \in \Lambda$ and some command $c \in \bar{\Lambda}\mu$.

**Lemma 15.** *For all $M, N \in \Lambda$, $\delta(M)[\delta(N)/x] \to^* \delta(M[N/x])$.*

*Proof.* By induction on the structure of $M \in \Lambda$.

- If $M = y \neq x$, then

$$\begin{aligned}\delta(y)[\delta(N)/x] &= (\mu\alpha.\langle y \parallel \alpha\rangle)[\delta(N)/x] \\ &= \mu\alpha.\langle y \parallel \alpha\rangle = \delta(y) = \delta(y[N/x])\end{aligned}$$

- If $M = x$, then

$$\begin{aligned}\delta(x)[\delta(N)/x] &= (\mu\alpha.\langle x \parallel \alpha\rangle)[\delta(N)/x] \\ &= \mu\alpha.\langle \delta(N) \parallel \alpha\rangle \\ &= \mu\alpha.\langle \mu\beta.c \parallel \alpha\rangle \text{ (by Fact 2)} \\ &\to_\mu \mu\alpha.c[\alpha/\beta] \\ &\equiv \mu\beta.c = \delta(N) = \delta(x[N/x])\end{aligned}$$

- If $M = \lambda y.M'$, then

$$\begin{aligned}\delta(M)[\delta(N)/x] &= (\mu\alpha.\langle \lambda y.\delta(M') \parallel \alpha\rangle)[\delta(N)/x] \\ &= \mu\alpha.\langle \lambda y.\delta(M')[\delta(N)/x] \parallel \alpha\rangle \\ &\to^* \mu\alpha.\langle \lambda y.\delta(M'[N/x]) \parallel \alpha\rangle \text{ (by the induction hypothesis)} \\ &= \delta(\lambda y.M'[N/x]) = \delta((\lambda y.M')[N/x])\end{aligned}$$

- If $M = M'N'$, then

$$\begin{aligned}\delta(M'N')[\delta(N)/x] &= (\mu\alpha.\langle \delta(M') \parallel \delta(N') \cdot \alpha\rangle)[\delta(N)/x] \\ &= \mu\alpha.\langle \delta(M')[\delta(N)/x] \parallel (\delta(N')[\delta(N)/x]) \cdot \alpha\rangle \\ &\to^* \mu\alpha.\langle \delta(M'[N/x]) \parallel \delta(N'[N/x]) \cdot \alpha\rangle \text{ (by the IH)} \\ &= \delta(M'[N/x]N'[N/x]) = \delta((M'N')[N/x])\end{aligned}$$

□

**Lemma 16.** *If $t \to t'$ then $\delta(t) \to^+ \delta(t')$, for all $t \in \Lambda$.*

*Proof.* By induction on the structure of $t$.

- If $t = x$, then $t \not\to$ and, hence, the lemma holds trivially.
- If $t = \lambda x.N$ and $t \to t'$, then we must have reduced $N \to N'$. The translation of $t$ is $\delta(\lambda x.N) = \mu\alpha.\langle \lambda x.\delta(N) \parallel \alpha\rangle$, for $\alpha$ fresh. By the induction hypothesis we get $\delta(N) \to^+ \delta(N')$ and, hence, $\delta(t) \to^+ \delta(t')$ as required.
- If $t = MN$ and $t \to t'$, then we have one of the following cases:
  - $M \to M'$ and $t' = M'N$. By the induction hypothesis we get $\delta(M) \to^+ \delta(M')$ and, hence, $\delta(t) = \mu\alpha.\langle \delta(M) \parallel \delta(N) \cdot \alpha\rangle \to^+ \mu\alpha.\langle \delta(M') \parallel \delta(N) \cdot \alpha\rangle = \delta(t')$.
  - $N \to N'$ and $t' = MN'$. By the induction hypothesis we get $\delta(N) \to^+ \delta(N')$ and, hence, $\delta(t) = \mu\alpha.\langle \delta(M) \parallel \delta(N) \cdot \alpha\rangle \to^+ \mu\alpha.\langle \delta(M) \parallel \delta(N') \cdot \alpha\rangle = \delta(t')$.

– $M = \lambda x.M'$, in which case $t = (\lambda x.M')N \to_\beta M'[N/x] = t'$. For the translation of $t$ we have:

$$\begin{aligned}
\delta(t) &= \mu\alpha.\langle \mu\beta.\langle \lambda x.\delta(M') \parallel \beta\rangle \parallel \delta(N)\cdot\alpha\rangle \\
&\to_\mu \mu\alpha.\langle \lambda x.\delta(M') \parallel \delta(N)\cdot\alpha\rangle \\
&\to_\lambda \mu\alpha.\langle \delta(M')[\delta(N)/x] \parallel \alpha\rangle \\
&\to^* \mu\alpha.\langle \delta(M'[N/x]) \parallel \alpha\rangle \text{ (by Lemma 15)} \\
&= \mu\alpha.\langle \mu\beta.c \parallel \alpha\rangle \text{ (by Fact 2)} \\
&\to_\mu \mu\alpha.c[\alpha/\beta] \equiv \mu\beta.c = \delta(M'[N/x]) = \delta(t')
\end{aligned}$$

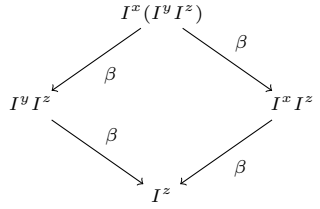
as required.

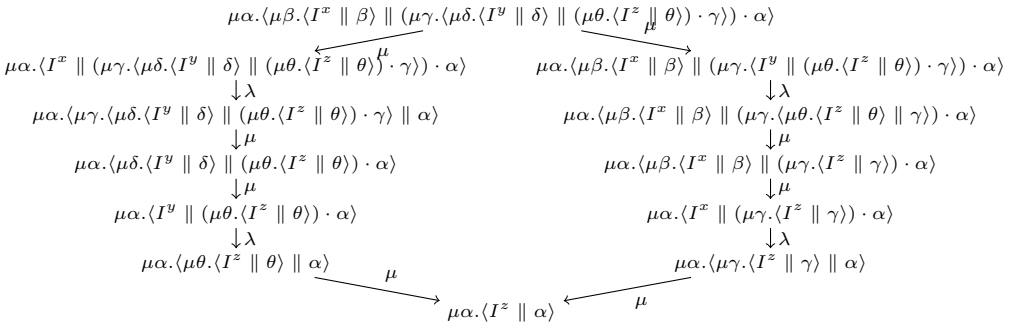
□

**Example 8.** Let us investigate how $t = I^x(I^y I^z)$ is reduced in $\bar{\lambda}\mu$, where we let $I^k$ denote the well-known identity function in the variable $k$ i.e. $I^k = \lambda k.k$. In a call-by-value calculus we must first reduce $I^y I^z$ to a value $v$ before we evaluate $I^x v$, hence we are forced to perform the following reduction:

$$t = I^x(I^y I^z) \to_{CBV} I^x I^z \to_{CBV} I^z$$

In the call-by-name calculus $\lambda$ we get the reduction paths in the first diagram below. Using the map $\delta$, we can also construct its $\bar{\lambda}\mu$ version.




*end of example*

**Lemma 17.** *The map $\delta$ preserves types. That is: if $\Gamma \vdash M : A$ in $\lambda_\to$, then $\Gamma \vdash \delta(M) : A \mid \emptyset$ in $\bar{\lambda}\mu$.*

*Proof.* By induction on the derivation of $\Gamma \vdash M : A$ in $\lambda_\to$.

AX We have $(x : A) \in \Gamma$ and we get the following derivation for $\delta(x) = \mu\alpha.\langle x \parallel \alpha\rangle$:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash x : A \mid \alpha : A}\ \text{AX}_r \quad \dfrac{}{\Gamma \mid \alpha : A \vdash \alpha : A}\ \text{AX}_l}{\langle x \parallel \alpha\rangle : (\Gamma \vdash \alpha : A)}\ \text{CM}}{\Gamma \vdash \mu\alpha.\langle x \parallel \alpha\rangle : A \mid \emptyset}\ \mu$$

$\to_I$ The translation of $\lambda x.M$ is $\mu\alpha.\langle \lambda x.\delta(M) \parallel \alpha\rangle$ and we get the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\textbf{IH}}{\Gamma, x : A \vdash \delta(M) : B \mid \emptyset}}{\Gamma, x : A \vdash \delta(M) : B \mid \alpha : A \to B}\ \text{WK}_{\Delta r}}{\Gamma \vdash \lambda x.\delta(M) : A \to B \mid \alpha : A \to B}\ \to_r \quad \dfrac{}{\Gamma \mid \alpha : A \to B \vdash \alpha : A \to B}\ \text{AX}_l}{\langle \lambda x.\delta(M) \parallel \alpha\rangle : (\Gamma \vdash \alpha : A \to B)}\ \text{CM}}{\Gamma \vdash \mu\alpha.\langle \lambda x.\delta(M) \parallel \alpha\rangle : A \to B \mid \emptyset}\ \mu$$

$\to_E$ The translation of $MN$ is $\mu\alpha.\langle \delta(M) \parallel \delta(N) \cdot \alpha\rangle$. We get the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\textbf{IH}}{\Gamma \vdash \delta(M) : A \to B \mid \emptyset}}{\Gamma \vdash \delta(M) : A \to B \mid \alpha : B}\ \text{WK}_{\Delta r} \quad \dfrac{\dfrac{\dfrac{\textbf{IH}}{\Gamma \vdash \delta(N) : A \mid \emptyset}}{\Gamma \vdash \delta(N) : A \mid \alpha : B}\ \text{WK}_{\Delta r} \quad \dfrac{}{\Gamma \mid \alpha : B \vdash \alpha : B}\ \text{AX}_r}{\Gamma \mid (\delta(N) \cdot \alpha) : A \to B \vdash \alpha : B}\ \to_l}{\langle \delta(M) \parallel \delta(N) \cdot \alpha\rangle : (\Gamma \vdash \alpha : B)}\ \text{CM}}{\Gamma \vdash \mu\alpha.\langle \delta(M) \parallel \delta(N) \cdot \alpha\rangle : B \mid \emptyset}\ \mu$$

The WK-rules are used to remove $\alpha : B$ from the context. We can safely do this, because $\alpha$ is a fresh (it does not occur free in $\delta(M)$ or $\delta(N)$) context-variable introduced by the map $\delta$.

□

## 3.1 An isomorphism

In [CH00], Curien and Herbelin define a map $\varphi : \Lambda\mu \to \bar{\Lambda}\mu$ as follows:

**Definition 19.**

$$\begin{aligned}
\varphi(x) &= x \\
\varphi(\lambda x.M) &= \lambda x.\varphi(M) \\
\varphi(MN) &= \mu\bar{\alpha}.\varphi(MN, \bar{\alpha}), \text{ for } \bar{\alpha} \text{ fresh} \\
\varphi(\mu\alpha.c) &= \mu\alpha.\varphi(c) \\
\varphi([\alpha]M) &= \varphi(M, \alpha)
\end{aligned}$$

$$\begin{aligned}
\varphi(MN, E) &= \varphi(M, \varphi(N) \cdot E) \\
\varphi(V, E) &= \langle \varphi(V) \parallel E\rangle, \text{ where } V = x \mid \lambda x.M \mid \mu\alpha.c
\end{aligned}$$

Our definition of $\varphi$ differs slightly from the one presented in [CH00] in terms of naming and notation. Note how $\mu$-abstractions are used to encode nested

applications as in $M(NP)$. We choose to mark the context-variables ($\bar{\alpha}$) over which we abstract in this case. This will make life a little easier later on, when we define the inverse of $\varphi$ and have to make a distinction between a "real" $\mu$-abstraction and one that was introduced by $\varphi$ to encode nested applications.

**Example 9.**

$$\begin{aligned}\varphi(x(yz)) &= \mu\bar{\alpha}.\varphi(x(yz),\bar{\alpha}) = \mu\bar{\alpha}.\varphi(x,\varphi(yz)\cdot\bar{\alpha})\\ &= \mu\bar{\alpha}.\varphi(x,(\mu\bar{\beta}.\varphi(yz,\bar{\beta}))\cdot\bar{\alpha})\\ &= \mu\bar{\alpha}.\varphi(x,(\mu\bar{\beta}.\varphi(y,\varphi(z)\cdot\bar{\beta}))\cdot\bar{\alpha})\\ &= \mu\bar{\alpha}.\langle\varphi(x)\parallel(\mu\bar{\beta}.\varphi(y,\varphi(z)\cdot\bar{\beta}))\cdot\bar{\alpha}\rangle\\ &= \mu\bar{\alpha}.\langle\varphi(x)\parallel(\mu\bar{\beta}.\langle\varphi(y)\parallel\varphi(z)\cdot\bar{\beta}\rangle)\cdot\bar{\alpha}\rangle\\ &= \mu\bar{\alpha}.\langle x\parallel(\mu\bar{\beta}.\langle y\parallel z\cdot\bar{\beta}\rangle)\cdot\bar{\alpha}\rangle\end{aligned}$$

*end of example*

In general we have the following equation for $\varphi$:

$\varphi(MN_1\ldots N_k) = \mu\bar{\alpha}.\langle\varphi(M)\parallel\varphi(N_1)\cdot\ldots\cdot\varphi(N_k)\cdot\bar{\alpha}\rangle$, where $M = x \mid \lambda x.N \mid \mu\alpha.c$

**Lemma 18.** *The map $\varphi:\Lambda\mu\to\bar{\Lambda}\mu$ maps normal terms to normal terms.*

*Proof.* See [CH00]. □

Furthermore it is noted in [CH00] that if we decompose $\bar{\lambda}\mu$'s $\mu$-reduction into:

$$\begin{array}{lrcl}(\mu_{app}) & \langle\mu\alpha.c\parallel v\cdot e\rangle & \to & \langle\mu\alpha.(c[v\cdot\alpha/\alpha])\parallel e\rangle\\ (\mu_{var}) & \langle\mu\alpha.c\parallel\beta\rangle & \to & c[\beta/\alpha]\end{array}$$

And restrict the syntax of $\lambda\mu$ such that applications in contexts like $\lambda x.[\,]$ and $M[\,]$ are disallowed. I.e. use the grammars in Definition 20 to generate terms (let us call the resulting system $\widehat{\lambda\mu}$).

**Definition 20.** The set $\widehat{\Lambda\mu}$ of $\widehat{\lambda\mu}$-terms is defined by the following grammars:

$$\begin{array}{lcl}v & ::= & x \mid \lambda x.v \mid \mu\alpha.c\\ c & ::= & [\beta]a\\ a & ::= & v \mid av\end{array}$$

**Remark 3.** Note that this restriction is not really a restriction, since any application $MN$ can be expanded to $\mu\alpha.[\alpha]MN$ for $\alpha$ fresh. So we can replace $M(NP)$ by $M(\mu\alpha.[\alpha]NP)$ and similarly $\lambda x.NP$ by $\lambda x.\mu\alpha.[\alpha]NP$. All applications in $\widehat{\lambda\mu}$ are now of the form $MN_1\cdots N_k = ((MN_1)\cdots)N_k$ with $M,N_1,\ldots,N_k = x \mid \lambda x.v \mid \mu\alpha.c$, which fits exactly with the grammar $a$ in Definition 20.

Then the restriction of $\varphi$ to $\widehat{\Lambda\mu}$ becomes an isomorphism [CH00]. That is: it is bijective, maps normal forms to normal forms and preserves reductions step by step. We will now check that this is indeed the case. The proof is twofold: first we define a map $\psi:\bar{\Lambda}\mu\to\widehat{\Lambda\mu}$ and show that it is the inverse of $\varphi$ (and vice-versa). Then we will show that both $\varphi$ and $\psi$ preserve reduction step by step.

**Definition 21.**

$$\begin{aligned}
\psi(x) &= x \\
\psi(\lambda x.v) &= \lambda x.\psi(v) \\
\psi(\mu\alpha.c) &= \mu\alpha.\psi(c) \\
\psi(\mu\bar{\alpha}.c) &= \psi(c) \\
\psi(\langle v \parallel \alpha\rangle) &= [\alpha]\psi(v) \\
\psi(\langle v \parallel \bar{\alpha}\rangle) &= \psi(v) \\
\psi(\langle v \parallel v' \cdot e\rangle) &= \psi(e, \psi(v)\psi(v'))
\end{aligned}$$

$$\begin{aligned}
\psi(\alpha, E) &= [\alpha]E \\
\psi(\bar{\alpha}, E) &= E \\
\psi(v \cdot e, E) &= \psi(e, E\psi(v))
\end{aligned}$$

This definition might look rather odd at first sight, because each time we encounter a context-variable we make a distinction on whether it is marked ($\bar{\alpha}$) or not ($\alpha$). The problem we attempt to solve with this is that $\mu$-abstractions play two roles: they are "normal" $\mu$-abstractions and they are used by the map $\varphi$ to encode (nested) applications. In the former case, the translation is what one would expect. In the latter case $\varphi$ translates any application $MN_1 \dots N_k$, where $M = x \mid \lambda x.v \mid \mu\alpha.c$, to $\mu\bar{\alpha}.\langle\varphi(M) \parallel \varphi(N_1) \cdot \ldots \cdot \varphi(N_k) \cdot \bar{\alpha}\rangle$. Now, for $\psi$ to be the inverse of $\varphi$, we must retrieve the original application $MN_1 \dots N_k$ i.e. we must "throw away" $\mu\bar{\alpha}.$ and $\bar{\alpha}$. This is exactly what we achieve in the definition above.

**Example 10.**

$$\begin{aligned}
\psi(\mu\bar{\alpha}.\langle y \parallel z \cdot \bar{\alpha}\rangle) &= \psi(\langle y \parallel z \cdot \bar{\alpha}\rangle) \\
&= \psi(\bar{\alpha}, \psi(y)\psi(z)) \\
&= \psi(y)\psi(z) = yz
\end{aligned}$$

*end of example*

In general we have the following equations for $\psi$:

$$\begin{aligned}
\psi(\langle v \parallel v_1 \cdot \ldots \cdot v_k \cdot \alpha\rangle) &= [\alpha](((\psi(v)\psi(v_1))\ldots)\psi(v_k)) \\
\psi(\langle v \parallel v_1 \cdot \ldots \cdot v_k \cdot \bar{\alpha}\rangle) &= ((\psi(v)\psi(v_1))\ldots)\psi(v_k)
\end{aligned}$$

**Lemma 19.** *$\varphi$ and $\psi$ are each others inverse. That is: $\psi(\varphi(t)) = t$, for all $t \in \widehat{\Lambda\mu}$, and $\varphi(\psi(t)) = t$, for all $t \in \bar{\Lambda}\mu$.*

*Proof.* Let $t \in \widehat{\Lambda\mu}$, we proceed by induction on the structure of $t$.

- If $t = x$, then $\psi(\varphi(x)) = \psi(x) = x$.
- If $t = \lambda x.M$, then $\psi(\varphi(\lambda x.M)) = \psi(\lambda x.\psi(M)) = \lambda x.\psi(\varphi(M))$. By the induction hypothesis we have $\psi(\varphi(M)) = M$ and, hence, $\psi(\varphi(\lambda x.M)) = \lambda x.M$ as required.

- If $t = \mu\alpha.c$, then $\psi(\varphi(\mu\alpha.c)) = \psi(\mu\alpha.\varphi(c)) = \mu\alpha.\psi(\varphi(c))$. By the induction hypothesis we have $\psi(\varphi(c)) = c$ and, hence, $\psi(\varphi(\mu\alpha.c)) = \mu\alpha.c$ as required.
- If $t = [\alpha]t'$, then $\psi(\varphi([\alpha]t')) = \psi(\varphi(t', \alpha))$. Now:
  - If $t' = x$ or $t' = \lambda x.M$ or $t' = \mu\beta.c$, then we get $\psi(\varphi(t', \alpha)) = \psi(\langle\varphi(t') \parallel \alpha\rangle) = [\alpha]\psi(\varphi(t'))$. By the induction hypothesis we get $\psi(\varphi(t')) = t'$ and, hence, $\psi(\varphi([\alpha]t')) = [\alpha]t'$ as required.
  - If $t'$ is an application i.e. $t' = MN_1 \cdots N_k$ with $M = x \mid \lambda x.v \mid \mu\alpha.c$, then

$$\begin{aligned}
\psi(\varphi(t', \alpha)) &= \psi(\varphi(MN_1 \cdots N_k, \alpha)) \\
&= \psi(\varphi(MN_1 \cdots N_{k-1}, \varphi(N_k) \cdot \alpha)) \\
&= \ldots = \psi(\varphi(M, \varphi(N_1) \cdots \varphi(N_k) \cdot \alpha)) \\
&= \psi(\langle\varphi(M) \parallel \varphi(N_1) \cdots \varphi(N_k) \cdot \alpha\rangle) \\
&= \psi(\varphi(N_2) \cdots \varphi(N_k) \cdot \alpha, \psi(\varphi(M))\psi(\varphi(N_1))) \\
&= \ldots = \psi(\alpha, \psi(\varphi(M))\psi(\varphi(N_1)) \cdots \psi(\varphi(N_k))) \\
&= [\alpha]\psi(\varphi(M))\psi(\varphi(N_1)) \cdots \psi(\varphi(N_k))
\end{aligned}$$

    By using the induction hypothesis on all $k$+1 terms we get $\psi(\varphi([\alpha]MN_1 \cdots N_k)) = [\alpha]MN_1 \cdots N_k$ as required.
- Similarly if $t = MN_1 \cdots N_k$, then

$$\begin{aligned}
\psi(\varphi(MN_1 \cdots N_k)) &= \psi(\mu\bar{\alpha}.\varphi(MN_1 \cdots N_k, \bar{\alpha})) \\
&= \psi(\mu\bar{\alpha}.\langle\varphi(M) \parallel \varphi(N_1) \cdots \varphi(N_k) \cdot \bar{\alpha}\rangle) \\
&= \psi(\langle\varphi(M) \parallel \varphi(N_1) \cdots \varphi(N_k) \cdot \bar{\alpha}\rangle) \\
&= \psi(\varphi(N_2) \cdots \varphi(N_k) \cdot \bar{\alpha}, \psi(\varphi(M))\psi(\varphi(N_1))) \\
&= \ldots = \psi(\bar{\alpha}, \psi(\varphi(M))\psi(\varphi(N_1)) \cdots \psi(\varphi(N_k))) \\
&= \psi(\varphi(M))\psi(\varphi(N_1)) \cdots \psi(\varphi(N_k))
\end{aligned}$$

  By using the induction hypothesis on all $k$+1 terms we get $\psi(\varphi(MN_1 \cdots N_k)) = MN_1 \cdots N_k$ as required.

The other way around we also have $\varphi(\psi(t)) = t$, for all $t \in \bar{\Lambda}\mu$. The proof is very similar to this one and is therefore omitted. It relies on the fact that marked context-variables are not originally in $\bar{\Lambda}\mu$ i.e. we can forget about the cases containing $\bar{\alpha}$ in the definition of $\psi$. □

We will now verify that $\varphi$ and $\psi$ preserve reduction step by step. But before we can do that, we need the following two lemmas.

**Lemma 20.** *Let $M, N \in \Lambda\mu$, then $\varphi(M[N/x]) = \varphi(M)[\varphi(N)/x]$.*

*Proof.* By induction on the structure of $M$.

- If $M = x$, then $\varphi(M[N/x]) = \varphi(N) = x[\varphi(N)/x] = \varphi(M)[\varphi(N)/x]$.
- If $M = \lambda y.M'$, then $\varphi(M[N/x]) = \lambda y.\varphi(M'[N/x]) = \lambda y.\varphi(M')[\varphi(N)/x]$ (by the induction hypothesis) $= (\lambda y.\varphi(M'))[\varphi(N)/x] = \varphi(M)[\varphi(N)/x]$.

- If $M = \mu\alpha.c$, then $\varphi(M[N/x]) = \mu\alpha.\varphi(c[N/x]) = \mu\alpha.\varphi(c)[\varphi(N)/x]$ (by the induction hypothesis) $= (\mu\alpha.\varphi(c))[\varphi(N)/x] = \varphi(M)[\varphi(N)/x]$.
- If $M = M'N_1 \dots N_k$ with $M' = x \mid \lambda x.N \mid \mu\alpha.c$, then $\varphi(M[N/x]) = \mu\bar{\alpha}.\langle\varphi(M'[N/x]) \parallel \varphi(N_1[N/x])\cdot\ldots\cdot\varphi(N_k[N/x])\cdot\bar{\alpha}\rangle = \mu\bar{\alpha}.\langle\varphi(M')[\varphi(N)/x] \parallel \varphi(N_1)[\varphi(N)/x] \cdot \ldots \cdot \varphi(N_k)[\varphi(N)/x] \cdot \bar{\alpha}\rangle$ (by the induction hypothesis) $= (\mu\bar{\alpha}.\langle\varphi(M') \parallel \varphi(N_1) \cdot \ldots \cdot \varphi(N_k) \cdot \bar{\alpha}\rangle)[\varphi(N)/x] = \varphi(M)[\varphi(N)/x]$.

□

**Lemma 21.** *Let $M, M', N \in \Lambda\mu$, then $\varphi(M[[\alpha]M'N/[\alpha]M']) = \varphi(M)[\varphi(N) \cdot \alpha/\alpha]$.*

*Proof.* By induction on the structure of $M$.

- If $M = x$, then $\varphi(M[[\alpha]M'N/[\alpha]M']) = \varphi(x) = x = x[\varphi(N) \cdot \alpha/\alpha] = \varphi(x)[\varphi(N) \cdot \alpha/\alpha] = \varphi(M)[\varphi(N) \cdot \alpha/\alpha]$.
- If $M = \lambda x.M_1$, then $\varphi(M[[\alpha]M'N/[\alpha]M']) = \lambda x.\varphi(M_1[[\alpha]M'N/[\alpha]M']) = \lambda x.\varphi(M_1)[\varphi(N)\cdot\alpha/\alpha]$ (by the induction hypothesis) $= (\lambda x.\varphi(M_1))[\varphi(N)\cdot\alpha/\alpha] = \varphi(M)[\varphi(N)\cdot\alpha/\alpha]$.
- If $M = \mu\beta.[\gamma]M_1$, then we have the following two cases:
  - If $\gamma = \alpha$, then $\varphi(M[[\alpha]M'N/[\alpha]M']) = \mu\beta.\varphi(([\gamma]M_1)[[\alpha]M'N/[\alpha]M']) = \mu\beta.\varphi([\gamma](M_1[[\alpha]M'N/[\alpha]M'])N) = \mu\beta.\langle\varphi(M_1)[\varphi(N)\cdot\alpha/\alpha] \parallel \varphi(N)\cdot\gamma\rangle$ (by the induction hypothesis) $= (\mu\beta.\langle\varphi(M_1) \parallel \gamma\rangle)[\varphi(N)\cdot\alpha/\alpha] = \varphi(M)[\varphi(N)\cdot\alpha/\alpha]$.
  - If $\gamma \neq \alpha$, then $\varphi(M[[\alpha]M'N/[\alpha]M']) = \mu\beta.\varphi([\gamma](M_1[[\alpha]M'N/[\alpha]M'])) = \mu\beta.\langle\varphi(M_1[[\alpha]M'N/[\alpha]M']) \parallel \gamma\rangle = \mu\beta.\langle\varphi(M_1)[\varphi(N)\cdot\alpha/\alpha] \parallel \gamma\rangle$ (by the induction hypothesis) $= (\mu\beta.\langle\varphi(M_1) \parallel \gamma\rangle)[\varphi(N)\cdot\alpha/\alpha] = \varphi(M)[\varphi(N)\cdot\alpha/\alpha]$.
- If $M = N_1N_2 \dots N_k$ with $N_i = x \mid \lambda x.M'' \mid \mu\beta.c$, then $\varphi(M[[\alpha]M'N/[\alpha]M']) = \mu\bar{\beta}.\langle\varphi(N_1[[\alpha]M'N/[\alpha]M']) \parallel \varphi(N_2[[\alpha]M'N/[\alpha]M'])\cdot\ldots\cdot\varphi(N_k[[\alpha]M'N/[\alpha]M'])\cdot\bar{\beta}\rangle = \mu\bar{\beta}.\langle\varphi(N_1)[\varphi(N)\cdot\alpha/\alpha] \parallel (\varphi(N_2)[\varphi(N)\cdot\alpha/\alpha]) \cdot \ldots \cdot (\varphi(N_k)[\varphi(N)\cdot\alpha/\alpha])\cdot\bar{\beta}\rangle$ (by the induction hypothesis) $= (\mu\bar{\beta}.\langle\varphi(N_1) \parallel \varphi(N_2)\cdot\ldots\cdot\varphi(N_k)\cdot\bar{\beta}\rangle)[\varphi(N)\cdot\alpha/\alpha] = \varphi(M)[\varphi(N)\cdot\alpha/\alpha]$.

□

**Corollary 3.** *Since $\varphi$ is an isomorphism (with inverse $\psi$), Lemma 20 and Lemma 21 also hold for $\psi$. That is: $\psi(t[v\cdot\alpha/\alpha]) = \psi(t)[[\alpha]M\psi(v)/[\alpha]M]$ and $\psi(t[t'/x]) = \psi(t)[\psi(t')/x]$, for $t, t' \in \bar{\Lambda}\mu$ and $M \in \Lambda\mu$.*

**Lemma 22.** *$\varphi$ preserves reduction step by step. That is: for all $t \in \Lambda\mu$, if $t \to t'$, then $\varphi(t) \to \varphi(t')$.*

*Proof.* By induction on the structure of $t$.

- If $t = x$, then $t \nrightarrow$.
- If $t = \lambda x.M$ and $t \to t'$, then $M \to M'$ and $t' = \lambda x.M'$. By the induction hypothesis we have $\varphi(M) \to \varphi(M')$ and, hence, $\varphi(\lambda x.M) = \lambda x.\varphi(M) \to \lambda x.\varphi(M') = \varphi(\lambda x.M')$.

- If $t = \mu\alpha.c$ and $t \to t'$, then $c \to c'$ and $t' = \mu\alpha.c'$. By the induction hypothesis we have $\varphi(c) \to \varphi(c')$ and, hence, $\varphi(\mu\alpha.c) = \mu\alpha.\varphi(c) \to \mu\alpha.\varphi(c') = \varphi(\mu\alpha.c')$.

- If $t = [\alpha](\mu\beta.c)$, then $t \to_{\mu R} c[\alpha/\beta] = t'$. We have: $\varphi([\alpha](\mu\beta.c)) = \langle\mu\beta.\varphi(c) \parallel \alpha\rangle \to_{\mu_{var}} \varphi(c)[\alpha/\beta] = \varphi(c[\alpha/\beta])$. Where we can safely do the last step because context-variables do not get translated by $\varphi$.

- If $t = N_1N_2 \dots N_k$, with $N_i = x \mid \lambda x.M \mid \mu\alpha.c$, and $t \to t'$, then $N_x \to N'_x$ for some $1 \leq x \leq k$. By the induction hypothesis we have $\varphi(N_x) \to \varphi(N'_x)$ and, hence, $\varphi(N_1N_2 \dots N_k) = \mu\bar{\alpha}.\langle\varphi(N_1) \parallel \varphi(N_2) \cdot \dots \cdot \varphi(N_k) \cdot \bar{\alpha}\rangle \to \mu\bar{\alpha}.\langle\varphi(N'_1) \parallel \varphi(N'_2) \cdot \dots \cdot \varphi(N'_k) \cdot \bar{\alpha}\rangle = \varphi(N'_1N'_2 \dots N'_k) = \varphi(t')$, where $N'_i = N_i$ if $i \neq x$.

- If $t = (\lambda x.M)N_1 \dots N_k$, then $t \to_\beta (M[N_1/x])N_2 \dots N_k = t'$ and we get: $\varphi((\lambda x.M)N_1 \dots N_k) = \mu\bar{\alpha}.\langle\lambda x.\varphi(M) \parallel \varphi(N_1) \cdot \dots \cdot \varphi(N_k) \cdot \bar{\alpha}\rangle \to_\lambda \mu\bar{\alpha}.\langle\varphi(M)[\varphi(N_1)/x] \parallel \varphi(N_2) \cdot \dots \cdot \varphi(N_k) \cdot \bar{\alpha}\rangle = \mu\bar{\alpha}.\langle\varphi(M[N_1/x]) \parallel \varphi(N_2) \cdot \dots \cdot \varphi(N_k) \cdot \bar{\alpha}\rangle$ (by Lemma 20) $= \varphi((M[N_1/x])N_2 \dots N_k)$.

- If $t = (\mu\alpha.c)N_1 \dots N_k$, then $t \to_\mu (\mu\alpha.c[[\alpha]MN_1/[\alpha]M])N_2 \dots N_k = t'$. First observe that the $\bar{\lambda}\mu$ equivalent of $c[[\alpha]MN_1/[\alpha]M]$ is $c[N_1 \cdot \alpha/\alpha]$. Then we get: $\varphi((\mu\alpha.c)N_1 \dots N_k) = \mu\bar{\alpha}.\langle\mu\alpha.\varphi(c) \parallel \varphi(N_1) \cdot \dots \cdot \varphi(N_k) \cdot \bar{\alpha}\rangle \to_{\mu_{app}} \mu\bar{\alpha}.\langle\mu\alpha.\varphi(c)[\varphi(N_1) \cdot \alpha/\alpha] \parallel \varphi(N_2) \cdot \dots \cdot \varphi(N_k) \cdot \bar{\alpha}\rangle = \mu\bar{\alpha}.\langle\mu\alpha.\varphi(c[N_1 \cdot \alpha/\alpha]) \parallel \varphi(N_2) \cdot \dots \cdot \varphi(N_k) \cdot \bar{\alpha}\rangle$ (by Lemma 21) $= \varphi((\mu\alpha.c[[\alpha]MN_1/[\alpha]M])N_2 \dots N_k)$.

□

**Lemma 23.** *$\psi$ preserves reduction step by step. That is: for all $t \in \bar{\Lambda}\mu$ if $t \to t'$, then $\psi(t) \to \psi(t')$.*

*Proof.* By induction on the structure of $t$.

- If $t = x$, then $t \nrightarrow$.

- If $t = \lambda x.v$ and $t \to t'$, then $v \to v'$. By the induction hypothesis we have $\psi(v) \to \psi(v')$ and, hence, $\psi(\lambda x.v) = \lambda x.\psi(v) \to \lambda x.\psi(v') = \psi(\lambda x.v') = \psi(t')$.

- If $t = \mu\alpha.c$ and $t \to t'$, then $c \to c'$. By the induction hypothesis we have $\psi(c) \to \psi(c')$ and, hence, $\psi(\mu\alpha.c) = \mu\alpha.\psi(c) \to \mu\alpha.\psi(c') = \psi(\mu\alpha.c') = \psi(t')$.

- If $t = \langle\lambda x.v_1 \parallel v_2 \cdot \dots \cdot v_k \cdot \alpha\rangle$, then $t \to_\lambda \langle v_1[v_2/x] \parallel v_3 \cdot \dots \cdot v_k \cdot \alpha\rangle = t'$ and $\psi(\langle\lambda x.v_1 \parallel v_2 \cdot \dots \cdot v_k \cdot \alpha\rangle) = [\alpha](((\lambda x.\psi(v_1))\psi(v_2)) \dots \psi(v_k)) \to_\beta [\alpha]((\psi(v_1)[\psi(v_2)/x])\psi(v_3) \dots \psi(v_k)) = [\alpha](\psi(v_1[v_2/x])\psi(v_3) \dots \psi(v_k))$ (by Lemma 20, Corollary 3) $= \psi(\langle v_1[v_2/x] \parallel v_3 \cdot \dots \cdot v_k \cdot \alpha\rangle)$.

- If $t = \langle\mu\alpha.c \parallel \beta\rangle$, then $t \to_{\mu_{var}} c[\beta/\alpha] = t'$ and $\psi(\langle\mu\alpha.c \parallel \beta\rangle) = [\beta](\mu\alpha.\psi(c)) \to_{\mu R} \psi(c)[\beta/\alpha] = \psi(c[\beta/\alpha])$. Where we can safely do the last step because context-variables do not get translated by $\psi$.

- If $t = \langle\mu\alpha.c \parallel v_1 \cdot \dots \cdot v_k \cdot \beta\rangle$, then $t \to_{\mu_{app}} \langle\mu\alpha.c[v_1 \cdot \alpha/\alpha] \parallel v_2 \cdot \dots \cdot v_k \cdot \beta\rangle = t'$ and $\psi(\langle\mu\alpha.c \parallel v_1 \cdot \dots \cdot v_k \cdot \beta\rangle) = [\beta]((\mu\alpha.\psi(c))\psi(v_1) \dots \psi(v_k)) \to_\mu [\beta]((\mu\alpha.\psi(c)[[\alpha]M\psi(v_1)/[\alpha]M])\psi(v_2) \dots \psi(v_k)) = [\beta]((\mu\alpha.\psi(c[v_1 \cdot \alpha/\alpha]))\psi(v_2) \dots \psi(v_k))$ (by Lemma 21, Corollary 3) $= \psi(\langle\mu\alpha.c[v_1 \cdot \alpha/\alpha] \parallel v_2 \cdot \dots \cdot v_k \cdot \beta\rangle)$.

□

Now that we have established that $\varphi$ is an isomorphism that preserves reduction, we will investigate whether both $\varphi$ and $\psi$ preserve types.

**Lemma 24.** *If* $\Gamma \vdash M : A \mid \Delta$ *in* $\lambda\mu$*, then* $\Gamma \vdash \varphi(M) : A \mid \Delta$ *in* $\bar{\lambda}\mu$*.*

*Proof.* By induction on the derivation of $\Gamma \vdash M : A \mid \Delta$ in $\lambda\mu$.

AX We have $(x : A) \in \Gamma$ and $\varphi(x) = x$ and, hence, $\Gamma \vdash \varphi(x) : A \mid \Delta$, by the $\text{AX}_r$-rule.

$\to_I$ By the induction hypothesis we get $\Gamma, x : A \vdash \varphi(M) : B \mid \Delta$. Then we get $\Gamma \vdash \lambda x.\varphi(M) : A \to B \mid \Delta$ by the $\to_r$-rule, as required since $\varphi(\lambda x.M) = \lambda x.\varphi(M)$.

$\mu$ By the induction hypothesis we get $\varphi(c) : (\Gamma \vdash \beta : B, \Delta)$ and, hence, $\Gamma \vdash \mu\beta.\varphi(c) : B \mid \Delta$ by the $\mu$-rule, as required.

$\to_E$ We get the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\textbf{IH}}{\Gamma \vdash \varphi(M) : A \to B \mid \Delta}}{\Gamma \vdash \varphi(M) : A \to B \mid \bar{\alpha} : B, \Delta}\ \text{WK} \quad \dfrac{\dfrac{\dfrac{\textbf{IH}}{\Gamma \vdash \varphi(N) : A \mid \Delta}}{\Gamma \vdash \varphi(N) : A \mid \bar{\alpha} : B, \Delta}\ \text{WK} \quad \dfrac{}{\Gamma \mid \bar{\alpha} : B \vdash \bar{\alpha} : B, \Delta}\ \text{AX}_l}{\Gamma \vdash (\varphi(N) \cdot \bar{\alpha}) : A \to B \mid \bar{\alpha} : B, \Delta}\ \to_l}{\langle \varphi(M) \parallel \varphi(N) \cdot \bar{\alpha} \rangle : (\Gamma \vdash \bar{\alpha} : B, \Delta)}\ \text{CM}}{\Gamma \vdash \mu\bar{\alpha}.\langle \varphi(M) \parallel \varphi(N) \cdot \bar{\alpha} \rangle : B \mid \Delta}\ \mu$$

The WK rules in the above derivation are used to remove $\bar{\alpha}$ out of the context. We can safely do this, because $\bar{\alpha}$ is a fresh context-variable (which is not free in both $\varphi(M)$ and $\varphi(N)$) introduced by the map $\varphi$.

CM $\varphi([\alpha]M) = \varphi(M, \alpha) = \langle \varphi(N) \parallel \varphi(N_1) \cdot \ldots \cdot \varphi(N_k) \cdot \alpha \rangle$, where $M = NN_1 \ldots N_k$ and $N = x \mid \lambda x.N' \mid \mu\alpha.c$.

$$\dfrac{\dfrac{\textbf{IH}}{\Gamma \vdash \varphi(N_1) : A_1 \mid \alpha : A, \Delta} \quad \dfrac{\dfrac{\dfrac{\textbf{IH}}{\Gamma \vdash \varphi(N_k) : A_k \mid \alpha : A, \Delta} \quad \dfrac{}{\Gamma \mid \alpha : A \vdash \alpha : A, \Delta}\ \text{AX}_l}{\Gamma \mid \varphi(N_k) \cdot \alpha : A_k \to A \vdash \alpha : A, \Delta}\ \to_l}{\Gamma \mid \varphi(N_1) \cdot \ldots \cdot \varphi(N_k) \cdot \alpha : A_1 \to \ldots \to A_k \to A \vdash \alpha : A, \Delta}\ \to_l}{\mathbf{T}}$$

$$\dfrac{\dfrac{\textbf{IH}}{\Gamma \vdash \varphi(N) : A_1 \to \ldots \to A_k \to A \mid \alpha : A, \Delta} \quad \mathbf{T}}{\langle \varphi(N) \parallel \varphi(N_1) \cdot \ldots \cdot \varphi(N_k) \cdot \alpha \rangle : (\Gamma \vdash \alpha : A, \Delta)}\ \text{CM}$$

□

**Example 11.** Consider, for example, the term $\lambda x.\mu\alpha.[\alpha]x(\lambda y.\mu\beta.[\alpha]y)$, whose type is Peirce's law in $\lambda\mu$. It's translation to $\bar{\lambda}\mu$ is:

$$\varphi(\lambda x.\mu\alpha.[\alpha]x(\lambda y.\mu\beta.[\alpha]y)) = \lambda x.\mu\alpha.\langle x \parallel (\lambda y.\mu\beta.\langle y \parallel \alpha \rangle) \cdot \alpha \rangle$$

and its type is again Peirce's law (to save space we do not write down types for terms in contexts):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x,y \vdash y : A \mid \beta, \alpha}\,\text{AX}_r \quad \dfrac{}{x,y \mid \alpha : A \vdash \beta, \alpha}\,\text{AX}_l}{\langle y \parallel \alpha\rangle : (x,y \vdash \beta, \alpha)}\,\text{CM}}{x, y \vdash \mu\beta.\langle y \parallel \alpha\rangle : B \mid \alpha}\,\mu}{x \vdash \lambda y.\mu\beta.\langle y \parallel \alpha\rangle : A \to B \mid \alpha}\,\to_r \qquad \dfrac{}{x \mid \alpha : A \vdash \alpha}\,\text{AX}_l}{x \mid (\lambda y.\mu\beta.\langle y \parallel \alpha\rangle)\cdot\alpha : (A \to B) \to A \vdash \alpha}\,\to_l}{\langle x \parallel (\lambda y.\mu\beta.\langle y \parallel \alpha\rangle)\cdot\alpha\rangle : (x \vdash \alpha)}\,\text{CM}}{x \vdash \mu\alpha.\langle x \parallel (\lambda y.\mu\beta.\langle y \parallel \alpha\rangle)\cdot\alpha\rangle : A \mid}\,\mu}{\vdash \lambda x.\mu\alpha.\langle x \parallel (\lambda y.\mu\beta.\langle y \parallel \alpha\rangle)\cdot\alpha\rangle : ((A \to B) \to A) \to A \mid}\,\to_r$$

*end of example*

**Fact 3.** It follows from the grammars in Definition 14 (definition of $\bar{\Lambda}\mu$) that, in $\bar{\lambda}\mu$, each context $e$ is of the form $v_1 \cdot \ldots \cdot v_k \cdot \alpha$. Now, if $\Gamma \mid e : A \vdash \Delta$, then $\Gamma \mid v_1 \cdot \ldots \cdot v_k \cdot \alpha : A \vdash \Delta$ and we must have that $A = A_1 \to \ldots \to A_k \to B$. Then, using the $\to_l$ typing rule, it easily follows that $\Gamma \mid \alpha : B \vdash \Delta$ and $\Gamma \vdash v_i : A_i \mid \Delta$, for $1 \leq i \leq k$.

**Lemma 25.** *If $\Gamma \vdash t : A \mid \Delta$ in $\bar{\lambda}\mu$, then $\Gamma \vdash \psi(t) : A \mid \Delta$ in $\lambda\mu$. Moreover if $c : (\Gamma \vdash \Delta)$ in $\bar{\lambda}\mu$, then $\psi(c) : (\Gamma \vdash \Delta)$ in $\lambda\mu$.*

*Proof.* By induction on $t \in \bar{\Lambda}\mu$.

- If $t = x$, then $\psi(t) = x$ and $(x : A) \in \Gamma$. Hence, $\Gamma \vdash \psi(x) : A \mid \Delta$ as required.
- If $t = \lambda x.v$, then $A = B \to C$ for some types $B$ and $C$ and $\Gamma, x : B \vdash v : C \mid \Delta$, by the $\to_r$-rule. We have $\psi(t) = \lambda x.\psi(v)$ and we get the following derivation in $\lambda\mu$:
$$\dfrac{\dfrac{\textbf{IH}}{\Gamma, x : B \vdash \psi(v) : C \mid \Delta}}{\Gamma \vdash \lambda x.\psi(v) : B \to C \mid \Delta}\,\to_I$$
- If $t = \mu\alpha.\langle v \parallel e\rangle$, then we get $\Gamma \vdash v : T \mid \alpha : A, \Delta$ and $\Gamma \mid e : T \vdash \alpha : A, \Delta$ by applying first the $\mu$ and then the CM-rule. By Fact 3, we must have that $T = A_1 \to \ldots \to A_k \to B$, since $e = v_1 \cdot \ldots \cdot v_k \cdot \beta$ (where possibly $\beta = \alpha$). Moreover $\Gamma \vdash v_i : A_i \mid \alpha : A, \Delta$ and $\Gamma \mid \beta : B \vdash \alpha : A, \Delta$ are derivable. For the translation $\psi(t) = \mu\alpha.[\beta](((\psi(v)\psi(v_1))\ldots)\psi(v_k))$ we get the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\textbf{IH}}{\Gamma \vdash v : A_1 \to \ldots \to A_k \to B \mid \alpha : A, \Delta} \quad \dfrac{\textbf{IH}}{\Gamma \vdash v_1 : A_1 \mid \alpha : A, \Delta}}{\Gamma \vdash \psi(v)\psi(v_1) : A_2 \to \ldots \to A_k \to B \mid \alpha : A, \Delta}\,\to_E}{\vdots}}{\Gamma \vdash ((\psi(v)\psi(v_1))\ldots)\psi(v_{k-1}) : A_k \to B \mid \alpha : A, \Delta}\,\to_E \qquad \dfrac{\textbf{IH}}{\Gamma \vdash v_k : A_k \mid \alpha : A, \Delta}}{\Gamma \vdash ((\psi(v)\psi(v_1))\ldots)\psi(v_k) : B \mid \alpha : A, \Delta}\,\to_E$$
$$\dfrac{\dfrac{\dfrac{\Gamma \vdash ((\psi(v)\psi(v_1))\ldots)\psi(v_k) : B \mid \alpha : A, \Delta}{([\beta](((\psi(v)\psi(v_1))\ldots)\psi(v_k))) : (\Gamma \vdash \alpha : A, \Delta)}\,\text{CM}}{\Gamma \vdash \mu\alpha.[\beta](((\psi(v)\psi(v_1))\ldots)\psi(v_k)) : A \mid \Delta}\,\mu$$

This lemma also holds for commands with $\langle v \parallel e\rangle : (\Gamma \vdash \Delta)$. We get $\Gamma \vdash v : T \mid \Delta$ and $\Gamma \mid e : T \vdash \Delta$ by the CM-rule. Using Fact 3 we know that

$e = v_1 \cdot \ldots \cdot v_k \cdot \beta$ and, hence, that $T = A_1 \to \ldots \to A_k \to B, \Gamma \vdash v_i : A_i \mid \Delta$ and $\Gamma \mid \beta : B \vdash \Delta$. For the translation $\psi(t) = [\beta](((\psi(v)\psi(v_1))\ldots)\psi(v_k))$ we get a derivation very similar to the one for $\mu$-abstractions.

□

We will now check that the $\mu_{app}$ and $\mu_{var}$ satisfy subject reduction. Before we can do that, however, we prove a substitution lemma slightly different from the one in Lemma 12. We will need this lemma in the subject reduction proof of the $\mu_{app}$ reduction.

**Lemma 26.** *If*

*(A)* $\Gamma \vdash v : A \mid \alpha : B \to C,\ \Delta$,

*(B)* $\Gamma \mid e : A \vdash \alpha : B \to C,\ \Delta$ *and*

*(C)* $\Gamma \vdash v' : B \mid \Delta$

*then the following statements hold:*

*(1)* $\Gamma \vdash v[v' \cdot \alpha/\alpha] : A \mid \alpha : C,\ \Delta$

*(2)* $\Gamma \mid e[v' \cdot \alpha/\alpha] : A \vdash \alpha : C,\ \Delta$

*Proof.* Simultaneously, by induction on $e, v$.

- (1) If $v = x$, then $v[v' \cdot \alpha/\alpha] = x$ and, hence, $\Gamma \vdash x : A \mid \alpha : C,\ \Delta$ by (A) and the fact that $\alpha$ was not cancelled in that proof.

  (2) If $e = \beta \neq \alpha$, then we get a similar argument as for $v = x$ above. If $e = \alpha$, then $e[v' \cdot \alpha/\alpha] = v' \cdot \alpha$ and we must have had that $A = B \to C$. Then we have the following derivation, as required:

$$\dfrac{\dfrac{\dfrac{(C)}{\Gamma \vdash v' : B \mid \Delta}}{\Gamma \vdash v' : B \mid \alpha : C,\ \Delta}\ \mathrm{WK}_{\Delta r}}{\Gamma \mid v' \cdot \alpha : B \to C \vdash \alpha : C,\ \Delta}\ \to_l$$

- (1) If $v = \lambda x.v''$, then $v[v' \cdot \alpha/\alpha] = \lambda x.v''[v' \cdot \alpha/\alpha]$ and $A = D \to E$. We get the following derivation:

$$\dfrac{\dfrac{\textbf{IH}}{\Gamma, /x : D \vdash v''[v' \cdot \alpha/\alpha] : E \mid \alpha : C,\ \Delta}}{\Gamma \vdash \lambda x.v''[v' \cdot \alpha/\alpha] : D \to E \mid \alpha : C,\ \Delta}\ \to_r$$

  (2) If $e = v'' \cdot e'$, then $e[v' \cdot \alpha/\alpha] = v''[v' \cdot \alpha/\alpha] \cdot e'[v' \cdot \alpha/\alpha]$ and $A = D \to E$. We get the following derivation:

$$\dfrac{\dfrac{\textbf{IH}}{\Gamma \vdash v''[v' \cdot \alpha/\alpha] : D \mid \alpha : C,\ \Delta} \quad \dfrac{\textbf{IH}}{\Gamma \mid e'[v' \cdot \alpha/\alpha] : E \vdash \alpha : C,\ \Delta}}{\Gamma \mid (v''[v' \cdot \alpha/\alpha] \cdot e'[v' \cdot \alpha/\alpha]) : D \to E \vdash \alpha : C,\ \Delta}\ \to_l$$

- If $v = \mu\beta.\langle v'' \parallel e'\rangle$ then $(\mu\beta.\langle v'' \parallel e'\rangle)[v' \cdot \alpha/\alpha] = \mu\beta.\langle v'' \parallel e'\rangle[v' \cdot \alpha/\alpha] = \mu\beta.\langle v''[v' \cdot \alpha/\alpha] \parallel e'[v' \cdot \alpha/\alpha]\rangle$. We get the following derivation, for some type $B$:

$$\dfrac{\dfrac{\dfrac{\textbf{IH}}{\Gamma \vdash v''[v' \cdot \alpha/\alpha] : B \mid \alpha : C,\ \Delta} \quad \dfrac{\textbf{IH}}{\Gamma \mid e'[v' \cdot \alpha/\alpha] : B \vdash \alpha : C,\ \Delta}}{\langle v''[v' \cdot \alpha/\alpha] \parallel e'[v' \cdot \alpha/\alpha]\rangle : (\Gamma \vdash \alpha : C,\ \Delta)}\ \text{CM}}{\Gamma \vdash \mu\beta.\langle v''[v' \cdot \alpha/\alpha] \parallel e'[v' \cdot \alpha/\alpha]\rangle : A \mid \alpha : C,\ \Delta}\ \mu$$

This lemma also holds for commands $\langle v \parallel e\rangle$, since $(\langle v \parallel e\rangle)[v' \cdot \alpha/\alpha] = \langle v[v' \cdot \alpha/\alpha] \parallel e[v' \cdot \alpha/\alpha]\rangle$. □

**Lemma 27.** *The $\mu_{app}$ and $\mu_{var}$ reductions satisfy subject reduction.*

*Proof.*

$\mu_{app}$ For some types $A$ and $B$ we have:

$$\dfrac{\dfrac{c : (\Gamma \vdash \alpha : A \to B,\ \Delta)}{\Gamma \vdash \mu\alpha.c : A \to B \mid \Delta}\ \mu \quad \dfrac{\Gamma \vdash v : A \mid \Delta \quad \Gamma \mid e : B \vdash \Delta}{\Gamma \mid (v \cdot e) : A \to B \vdash \Delta}\ \to_l}{\langle \mu\alpha.c \parallel v \cdot e\rangle : (\Gamma \vdash \Delta)}\ \text{CM}$$

$$\longrightarrow_{\mu_{app}}$$

$$\dfrac{\dfrac{\dfrac{\text{Lemma 26}}{c[v \cdot \alpha/\alpha] : (\Gamma \vdash \alpha : B,\ \Delta)}}{\Gamma \vdash \mu\alpha.c[v \cdot \alpha/\alpha] : B \mid \Delta}\ \mu \quad \Gamma \mid e : B \vdash \Delta}{\langle \mu\alpha.c[v \cdot \alpha/\alpha] \parallel e\rangle : (\Gamma \vdash \Delta)}\ \text{CM}$$

$\mu_{var}$ For some type $A$ we have:

$$\dfrac{\dfrac{c : (\Gamma \vdash \alpha : A,\ \Delta)}{\Gamma \vdash \mu\alpha.c : A \mid \Delta}\ \mu \quad \Gamma \mid \beta : A \vdash \Delta}{\langle \mu\alpha.c \parallel \beta\rangle : (\Gamma \vdash \Delta)}\ \text{CM}$$

Then we get $c[\beta/\alpha]$ by Corollary 1.

□

We will now show that the reductions in $\bar{\lambda}\mu$, where the $\mu$-reduction is split into $\mu_{app}$ and $\mu_{var}$, are confluent. To this end we first need a lemma to establish closedness of $\widehat{\Lambda\mu}$.

**Lemma 28.** *$\widehat{\Lambda\mu}$ is closed under reduction. That is: for all $a \in \widehat{\Lambda\mu}$, if $a \to a'$ then $a' \in \widehat{\Lambda\mu}$.*

*Proof.* By induction on the structure of $a \in \widehat{\Lambda\mu}$.

- If $a = x$, then $a \nrightarrow$ and the lemma holds trivially.
- If $a = \lambda x.v$, then we must have reduced $v \to v'$. By the induction hypothesis we have $v' \in \widehat{\Lambda\mu}$ and, hence, $a = \lambda x.v \to \lambda x.v' = a' \in \widehat{\Lambda\mu}$ as required.

- If $a = \mu\alpha.[\beta]a'$, then we have the following two cases:
  - Either $a' = \mu\gamma.c$, in which case $a = \mu\alpha.[\beta](\mu\gamma.c) \to_{\mu R} \mu\alpha.c[\beta/\gamma]$. Now $\mu\alpha.c[\beta/\gamma] \in \widehat{\Lambda\mu}$ since $c \in \widehat{\Lambda\mu}$ and renaming does not change the shape of $c$.
  - Or $a' = vv_1 \dots v_n$, in which case we must have reduced $a' = vv_1 \dots v_n \to a''$. By the induction hypothesis we have that $a'' \in \widehat{\Lambda\mu}$ and, hence, $\mu\alpha.[\beta]a'' \in \widehat{\Lambda\mu}$ as required.
- If $a = vv_1 \dots v_n$, then we have the following two cases:
  - $v = \lambda x.v'$ in which case $a = (\lambda x.v')v_1 \dots v_n \to_\beta (v'[v_1/x])v_2 \dots v_n = a'$. Now we also have $a' \in \widehat{\Lambda\mu}$, as it can be shown by an easy induction that $v'[v_1/x] \in \widehat{\Lambda\mu}$.
  - $v = \mu\alpha.c$ in which case $a = (\mu\alpha.c)v_1 \dots v_n \to_\mu (\mu\alpha.c[[\alpha]a''v_1/[\alpha]a''])v_2 \dots v_n = a'$. Now $a' \in \widehat{\Lambda\mu}$ as it could be shown by an easy induction that $c[[\alpha]a''v_1/[\alpha]a''] \in \widehat{\Lambda\mu}$. The intuition is that already $c \in \widehat{\Lambda\mu}$ and $[\alpha]a'' \in \widehat{\Lambda\mu}$, by the induction hypothesis. But also $[\alpha]a''v_1 \in \widehat{\Lambda\mu}$ and, hence, $c[[\alpha]a''v_1/[\alpha]a''] \in \widehat{\Lambda\mu}$ as required.

□

**Lemma 29.** *The reductions in $\bar{\lambda}\mu$ (where the $\mu$-reduction is split into $\mu_{app}$ and $\mu_{var}$) are confluent.*

*Proof.* By Lemma 28 we have that $\widehat{\Lambda\mu}$ is closed under reduction. The reductions in $\widehat{\lambda\mu}$ are confluent since these reductions are the ones of $\lambda\mu$, which are confluent [Par92]. We obtain confluence for $\bar{\lambda}\mu$ by the $\varphi$-isomorphism. □

## 3.2 Exception handling

In $\bar{\lambda}\mu$ we can define an exception handling mechanism similar to that for $\lambda\mu$, which is defined in [Kre10].

**Definition 22.** We define the control operators `catch` and `throw` as folows:

$$\texttt{catch}_\alpha\ v = \mu\alpha.\langle v \parallel \alpha\rangle$$
$$\texttt{throw}_\beta\ v = \mu\gamma.\langle v \parallel \beta\rangle \quad \text{, where } \gamma \notin FV(t)$$

**Lemma 30.** *We have the following reductions for `catch` and `throw` in $\bar{\lambda}\mu$:*

1. $\mu\alpha.\langle \texttt{throw}_\beta\ v \parallel e\rangle \to \texttt{throw}_\beta\ v$, *provided that* $\alpha \notin FV(v)$
2. $\texttt{catch}_\alpha\ (\texttt{throw}_\alpha\ v) \to \texttt{catch}_\alpha\ v$
3. $\texttt{catch}_\alpha\ (\texttt{throw}_\beta\ v) \to \texttt{throw}_\beta\ v$, *provided that* $\alpha \notin FV(v)$
4. $\texttt{throw}_\beta\ (\texttt{throw}_\alpha\ v) \to \texttt{throw}_\alpha\ v$

*Proof.*

1. $\mu\alpha.\langle \texttt{throw}_\beta\ v \parallel e\rangle = \mu\alpha.\langle \mu\gamma.\langle v \parallel \beta\rangle \parallel e\rangle \to_\mu \mu\alpha.\langle v \parallel \beta\rangle = \texttt{throw}_\beta\ v$, since $\alpha, \gamma \notin FV(v)$.

2. $\texttt{catch}_\alpha\ (\texttt{throw}_\alpha\ v) = \mu\alpha.\langle\mu\beta.\langle v \parallel \alpha\rangle \parallel \alpha\rangle \rightarrow_\mu \mu\alpha.\langle v \parallel \alpha\rangle = \texttt{catch}_\alpha\ v.$
3. $\texttt{catch}_\alpha\ (\texttt{throw}_\beta\ v) = \mu\alpha.\langle\mu\gamma.\langle v \parallel \beta\rangle \parallel \alpha\rangle \rightarrow_\mu \mu\alpha.\langle v \parallel \beta\rangle = \texttt{throw}_\beta\ v$, since $\alpha, \gamma \notin FV(v)$.
4. $\texttt{throw}_\beta\ (\texttt{throw}_\alpha\ v) = \mu\gamma.\langle\mu\delta.\langle v \parallel \alpha\rangle \parallel \beta\rangle \rightarrow_\mu \mu\gamma.\langle v \parallel \alpha\rangle = \texttt{throw}_\alpha\ v$

□

While the intuition of rules (3)-(4) should be clear, we needs some explanation for rule (1). Rule (1) states that, if a `throw` occurs in a function position (the left side of a command), then this `throw` can be propagated upwards for as long as the context variable over which we abstract does not occur in the body of the throw. Consider the following example, which shows a case in which a throw cannot be propagated upwards:

**Example 12.** If we let $v = \mu\delta.\langle\lambda x.x \parallel \alpha\rangle$ in $\mu\alpha.\langle\texttt{throw}_\beta\ v \parallel \lambda y.y \cdot \alpha\rangle$, then:

$$\begin{aligned}\mu\alpha.\langle\texttt{throw}_\beta\ v \parallel \lambda y.y \cdot \alpha\rangle &= \mu\alpha.\langle\mu\gamma.\langle\mu\delta.\langle\lambda x.x \parallel \alpha\rangle \parallel \beta\rangle \parallel \lambda y.y \cdot \alpha\rangle \\ &\rightarrow \mu\alpha.\langle\mu\delta.\langle\lambda x.x \parallel \alpha\rangle \parallel \beta\rangle \\ &\neq \texttt{throw}_\beta\ v \quad \text{, since } \alpha \in FV(v)\end{aligned}$$

*end of example*

One might wonder whether or not it is possible to have a reduction similar to (1), but now for a `throw` in an argument position i.e. have the following reduction:

$$\mu\alpha.\langle\lambda x.v_1 \parallel (\texttt{throw}_\beta\ v_2) \cdot e\rangle \rightarrow \texttt{throw}_\beta\ v_2 \text{ , provided that } \alpha \notin FV(v_2)$$

Unfortunately though, we cannot expect such a reduction to hold in any call-by-name reduction system. The problem is that the `throw` will be moved into the body of $v_1$ and then anything can happen. Consider the following example, which shows that it is even possible to throw away a `throw` in such position completely:

**Example 13.** If we let $v_1 = \lambda x.\lambda y.y$ in $\mu\alpha.\langle v_1 \parallel (\texttt{throw}_\beta\ v_2) \cdot e\rangle$, then:

$$\begin{aligned}\mu\alpha.\langle v_1 \parallel (\texttt{throw}_\beta\ v_2) \cdot e\rangle &= \mu\alpha.\langle\lambda x.\lambda y.y \parallel (\texttt{throw}_\beta\ v_2) \cdot e\rangle \\ &\rightarrow_\lambda \mu\alpha.\langle\lambda y.y \parallel e\rangle\end{aligned}$$

*end of example*

## 3.3 An environment machine for $\bar{\lambda}\mu$

It is the task of an abstract machine to calculate the weak head normal form of a term in the calculus it is defined on. Such machine can also be seen as the operational semantics of the calculus, since it explains the meaning of a term in terms of simple stack and/or environment manipulations.

Herbelin and Curien define an abstract (environment) machine for the $\bar{\lambda}\mu\tilde{\mu}$ calculus in [CH00], which we will study later on. In this section we will first introduce the notion of weak head reduction in $\bar{\lambda}\mu$. Then we will show that a restriction of Herbelin and Curiens machine to $\bar{\lambda}\mu$ is sound and complete for this notion of reduction.

**Definition 23.** The weak head reduction strategy for $\bar{\lambda}\mu$ is defined by the following rules:

- $\langle \lambda x.v \parallel v' \cdot e \rangle \to_{wh} \langle v[v'/x] \parallel e \rangle$
- $\langle \mu\alpha.c \parallel e \rangle \to_{wh} c[e/\alpha]$
- $t \to_{wh}^* t$
- $\dfrac{c \to_{wh}^* c'}{\mu\alpha.c \to_{wh}^* \mu\alpha.c'}$

The abstract machine uses machine states of the form $\langle v\{E_1\} \parallel e\{E_2\} \rangle$. Here, $E_1$ and $E_2$ denote environments, which are lists of bindings of the form $x \mapsto v\{E\}$, where $E$ is again an environment. We write $E(x) = v\{E'\}$ if $E$ contains $x \mapsto v\{E'\}$ and let $\{\}$ denote the empty environment.

**Definition 24.** The environment machine for $\bar{\lambda}\mu$, which is a restriction of the one presented in [CH00] for $\bar{\lambda}\mu\tilde{\mu}$, is defined by the following reductions:

1. $\langle x\{E_1\} \parallel e\{E_2\} \rangle \to \langle E_1(x) \parallel e\{E_2\} \rangle$ (if $E_1(x)$ is defined)
2. $\langle v\{E_1\} \parallel \alpha\{E_2\} \rangle \to \langle v\{E_1\} \parallel E_2(\alpha) \rangle$ (if $E_2(\alpha)$ is defined)
3. $\langle (\lambda x.v_1)\{E_1\} \parallel (v_2 \cdot e)\{E_2\} \rangle \to \langle v_1\{x \mapsto v_2\{E_2\}, E_1\} \parallel e\{E_2\} \rangle$
4. $\langle \mu\alpha.\langle v\{E_1\} \parallel e_2\{E_2\} \rangle \parallel e_1\{E_3\} \rangle \to \langle v\{\alpha \mapsto e_1\{E_3\}, E_1\} \parallel e_2\{\alpha \mapsto e_1\{E_3\}, E_2\} \rangle$

This machine is non-deterministic. The machine states $\langle x\{E_1\} \parallel \alpha\{E_2\} \rangle$ and $\langle \mu\alpha.\langle v\{E_1\} \parallel e\{E_2\} \rangle \parallel \beta\{E_3\} \rangle$ can both be evaluated in two ways. The following lemma shows that, in these cases, we can always find a common "reduct".

**Lemma 31.** *The machine's reductions are locally confluent. That is: if $\mathcal{M}$ is a machine state and $\mathcal{M} \to \mathcal{M}_1$ and $\mathcal{M} \to \mathcal{M}_2$, then there exists a state $\mathcal{M}'$ such that $\mathcal{M}_1 \to^* \mathcal{M}'$ and $\mathcal{M}_2 \to^* \mathcal{M}'$.*

*Proof.* We must check two cases:

- If $\mathcal{M} = \langle x\{E_1\} \parallel \alpha\{E_2\} \rangle$ and $E_1(x)$ and $E_2(\alpha)$ are defined, then $\mathcal{M} \to_1 \langle E_1(x) \parallel \alpha\{E_2\} \rangle = \mathcal{M}_1$ and $\mathcal{M} \to_2 \langle x\{E_1\} \parallel E_2(\alpha) \rangle = \mathcal{M}_2$. Now if we take $\mathcal{M}' = \langle E_1(x) \parallel E_2(\alpha) \rangle$, then $\mathcal{M}_1 \to_2 \mathcal{M}'$ and $\mathcal{M}_2 \to_1 \mathcal{M}'$, as required.
- If $\mathcal{M} = \langle \mu\beta.\langle v\{E_1\} \parallel e\{E_2\} \rangle \parallel \alpha\{E_3\} \rangle$ and $E_3(\alpha)$ is defined, then $\mathcal{M} \to_2 \langle \mu\beta.\langle v\{E_1\} \parallel e\{E_2\} \rangle \parallel E_3(\alpha) \rangle = \mathcal{M}_1$ and $\mathcal{M} \to_4 \langle v\{\beta \mapsto \alpha\{E_3\}, E_1\} \parallel e\{\beta \mapsto \alpha\{E_3\}, E_2\} \rangle = \mathcal{M}_2$. Note that $\beta \mapsto \alpha\{E_3\} = \beta \mapsto E_3(\alpha)$, since $E_3(\alpha)$ is defined. Now if we take $\mathcal{M}' = \langle v\{\beta \mapsto E_3(\alpha), E_1\} \parallel e\{\beta \mapsto E_3(\alpha), E_2\} \rangle$, then $\mathcal{M}_1 \to_4 \mathcal{M}'$ and $\mathcal{M}_2 = \mathcal{M}'$, as required.

□

In principle, we can only evaluate commands with this machine. This does, however, make sense, since commands $\langle v \parallel e \rangle$ correspond to "complete" programs (it is the context $e$ whose hole is filled with $v$). Also note that we can never evaluate under a $\lambda$-abstraction, which is what we want. We do, however, want to evaluate under $\mu$-abstractions, since applications are necessarily encoded by a term-context pair (a command).

Evaluation of a term $t \in \bar{\Lambda}\mu$ starts in the machine state $t\{\}$. Indeed, if $t \neq \mu\alpha.c$ and $t \neq \langle v \parallel e \rangle$, then the machine immediately halts, as required. Moreover, if $t = \mu\alpha.\langle v \parallel e \rangle$, then $(\mu\alpha.\langle v \parallel e \rangle)\{\} = \mu\alpha.\langle v\{\} \parallel e\{\} \rangle$ and if $t = \langle v \parallel e \rangle$, then $\langle v \parallel e \rangle\{\} = \langle v\{\} \parallel e\{\} \rangle$. At the top-level, we may apply the following equation:

$$\langle (\mu\alpha.\langle v \parallel e \rangle)\{E_1\} \parallel e'\{E_2\} \rangle = \langle \mu\alpha.\langle v\{E_1\} \parallel e\{E_1\} \rangle \parallel e'\{E_2\} \rangle$$

**Example 14.** Suppose we want to evaluate the term $t = \mu\alpha.\langle \mu\beta.\langle \lambda x.x \parallel (\lambda y.y) \cdot \beta \rangle \parallel (\lambda z.z) \cdot \alpha \rangle$ (which is the $\bar{\lambda}\mu$ equivalent of the application $((\lambda x.x)\lambda y.y)\lambda z.z$ from the "ordinary" $\lambda$-calculus). Then we get the following reduction sequence:

$$\begin{aligned}
&\mu\alpha.\langle \mu\beta.\langle (\lambda x.x)\{\} \parallel ((\lambda y.y) \cdot \beta)\{\} \rangle \parallel ((\lambda z.z) \cdot \alpha)\{\} \rangle \\
&\to \mu\alpha.\langle (\lambda x.x)\{\beta \mapsto ((\lambda z.z) \cdot \alpha)\{\}\} \parallel ((\lambda y.y) \cdot \beta)\{\beta \mapsto ((\lambda z.z) \cdot \alpha)\{\}\} \rangle \\
&\to \mu\alpha.\langle x\{x \mapsto (\lambda y.y)\{\beta \mapsto ((\lambda z.z) \cdot \alpha)\{\}\}, \beta \mapsto ((\lambda z.z) \cdot \alpha)\{\}\} \parallel \beta\{\beta \mapsto ((\lambda z.z) \cdot \alpha)\{\}\} \rangle \\
&\to \mu\alpha.\langle x\{x \mapsto (\lambda y.y)\{\beta \mapsto ((\lambda z.z) \cdot \alpha)\{\}\}, \beta \mapsto ((\lambda z.z) \cdot \alpha)\{\}\} \parallel ((\lambda z.z) \cdot \alpha)\{\} \rangle \\
&\to \mu\alpha.\langle (\lambda y.y)\{\beta \mapsto ((\lambda z.z) \cdot \alpha)\{\}\} \parallel ((\lambda z.z) \cdot \alpha)\{\} \rangle \\
&\to \mu\alpha.\langle y\{y \mapsto (\lambda z.z)\{\}, \beta \mapsto ((\lambda z.z) \cdot \alpha)\{\}\} \parallel \alpha\{\} \rangle \\
&\to \mu\alpha.\langle (\lambda z.z)\{\} \parallel \alpha\{\} \rangle
\end{aligned}$$

Using the weak head reduction rules from Definition 23, it can be easily checked that $\mu\alpha.\langle \lambda z.z \parallel \alpha \rangle$ is indeed the weak head normal form of $t$.

*end of example*

Indeed, the abstract machine correctly implements the weak head reduction strategy from Definition 23. To prove this we first define a translation that transforms a term with environment $t\{E\}$ into a term $[\![t\{E\}]\!] \in \bar{\Lambda}\mu$. Then, we will show that the machine is correct for its weak head reduction strategy i.e. that the following diagram commutes, where $\mathcal{M}$ and $\mathcal{M}'$ are machine states:

$$\begin{array}{ccc}
\mathcal{M} & \longrightarrow & \mathcal{M}' \\
\Big\downarrow{\scriptstyle [\![-]\!]} & & \Big\downarrow{\scriptstyle [\![-]\!]} \\
[\![\mathcal{M}]\!] & \xrightarrow{wh^*} & [\![\mathcal{M}']\!]
\end{array}$$

**Definition 25.** The map $[\![-]\!] : \mathbf{S} \to \bar{\Lambda}\mu$, where $\mathbf{S}$ is the category of machine states, is defined as follows. We use $\nu$ to denote either a term-variable or a context-variable (i.e. $\nu \in \mathcal{X}$ or $\nu \in \mathcal{A}$)

$$[\![t\{E\}]\!] = \textsc{subs}(t, \{E\})$$

$$\begin{aligned}
\textsc{subs}(t, \{\}) &= t \\
\textsc{subs}(t, \{E, \nu \mapsto t'\{E'\}\}) &= \textsc{subs}(t[\textsc{subs}(t', \{E'\})/\nu], \{E\})
\end{aligned}$$

**Lemma 32.** *The machine is correct for the weak head reduction strategy defined in Definition 23. Let $\mathcal{M}$ be a machine state. If $\mathcal{M} \to_{1,2} \mathcal{M}'$, then $[\![\mathcal{M}]\!] = [\![\mathcal{M}']\!]$. If $\mathcal{M} \to_{3,4} \mathcal{M}'$, then $[\![\mathcal{M}]\!] \to_{wh} [\![\mathcal{M}']\!]$.*

*Proof.*

- If $\mathcal{M} = \langle x\{E_1\} \parallel e\{E_2\}\rangle$, then $\mathcal{M}' = \langle E_1(x) \parallel e\{E_2\}\rangle$ (if $E_1(x)$ is defined i.e. $E_1(x) = v\{E\}$ for some $v$ and $E$) and:

$$\begin{aligned}[\![\mathcal{M}]\!] &= [\![\langle x\{E_1\} \parallel e\{E_2\}\rangle]\!] \\ &= \langle \textsc{subs}(x, \{E_1\}) \parallel \textsc{subs}(e, \{E_2\})\rangle \\ &= \langle \textsc{subs}(v, \{E\}) \parallel \textsc{subs}(e, \{E_2\})\rangle \\ &= [\![\langle v\{E\} \parallel e\{E_2\}\rangle]\!] = [\![\mathcal{M}']\!]\end{aligned}$$

- If $\mathcal{M} = \langle v\{E_1\} \parallel \alpha\{E_2\}\rangle$, then $\mathcal{M}' = \langle v \parallel E_2(\alpha)\rangle$ (if $E_2(\alpha)$ is defined i.e. $E_2(\alpha) = e\{E\}$ for some $e$ and $E$) and:

$$\begin{aligned}[\![\mathcal{M}]\!] &= [\![\langle v\{E_1\} \parallel \alpha\{E_2\}\rangle]\!] \\ &= \langle \textsc{subs}(v, \{E_1\}) \parallel \textsc{subs}(\alpha, \{E_2\})\rangle \\ &= \langle \textsc{subs}(v, \{E_1\}) \parallel \textsc{subs}(e, \{E\})\rangle \\ &= [\![\langle v\{E_1\} \parallel e\{E\}\rangle]\!] = [\![\mathcal{M}']\!]\end{aligned}$$

- If $\mathcal{M} = \langle (\lambda x.v_1)\{E_1\} \parallel (v_2 \cdot e)\{E_2\}\rangle$, then $\mathcal{M}' = \langle v_1\{x \mapsto v_2\{E_2\}, E_1\} \parallel e\{E_2\}\rangle$ and:

$$\begin{aligned}[\![\mathcal{M}]\!] &= [\![\langle (\lambda x.v_1)\{E_1\} \parallel (v_2 \cdot e)\{E_2\}\rangle]\!] \\ &= \langle \textsc{subs}(\lambda x.v_1, \{E_1\}) \parallel \textsc{subs}(v_2 \cdot e, \{E_2\})\rangle \\ &= \langle \lambda x.\textsc{subs}(v_1, \{E_1\}) \parallel \textsc{subs}(v_2, \{E_2\}) \cdot \textsc{subs}(e, \{E_2\})\rangle \\ &\to_{wh} \langle \textsc{subs}(v_1, \{E_1\})[\textsc{subs}(v_2, \{E_2\})/x] \parallel \textsc{subs}(e, \{E_2\})\rangle \\ &= \langle \textsc{subs}(\textsc{subs}(v_1, \{E_1\}), \{x \mapsto v_2\{E_2\}\}) \parallel \textsc{subs}(e, \{E_2\})\rangle \\ &= \langle \textsc{subs}(v_1, \{x \mapsto v_2\{E_2\}, E_1\}) \parallel \textsc{subs}(e, \{E_2\})\rangle \\ &= [\![\langle v_1\{x \mapsto v_2\{E_2\}, E_1\} \parallel e\{E_2\}\rangle]\!] = [\![\mathcal{M}']\!]\end{aligned}$$

- If $\mathcal{M} = \langle \mu\alpha.\langle v\{E_1\} \parallel e_1\{E_2\}\rangle \parallel e_2\{E_3\}\rangle$, then $\mathcal{M}' = \langle v\{\alpha \mapsto e_2\{E_3\}, E_1\} \parallel e_1\{\alpha \mapsto e_2\{E_3\}, E_2\}\rangle$ and:

$$\begin{aligned}[\![\mathcal{M}]\!] &= [\![\langle \mu\alpha.\langle v\{E_1\} \parallel e_1\{E_2\}\rangle \parallel e_2\{E_3\}\rangle]\!] \\ &= \langle \mu\alpha.\langle \textsc{subs}(v, E_1) \parallel \textsc{subs}(e_1, E_2)\rangle \parallel \textsc{subs}(e_2, E_3)\rangle \\ &\to_{wh} \langle \textsc{subs}(v, E_1)[\textsc{subs}(e_2, E_3)/\alpha] \parallel \textsc{subs}(e_1, E_2)[\textsc{subs}(e_2, E_3)/\alpha]\rangle \\ &= \langle \textsc{subs}(\textsc{subs}(v, E_1), \{\alpha \mapsto e_2, E_3\}) \parallel \textsc{subs}(\textsc{subs}(e_1, E_2), \{\alpha \mapsto e_2\{E_3\}\})\rangle \\ &= \langle \textsc{subs}(v, \{\alpha \mapsto e_2\{E_3\}, E_1\}) \parallel \textsc{subs}(e_1, \{\alpha \mapsto e_2\{E_3\}, E_2\})\rangle \\ &= [\![\langle v\{\alpha \mapsto e_2\{E_3\}, E_1\} \parallel e_1\{\alpha \mapsto e_2\{E_3\}, E_2\}\rangle]\!] = [\![\mathcal{M}']\!]\end{aligned}$$

□

This lemma does not hold the other way around! For arbitrary $\bar{\lambda}\mu$ terms we simply do not know in which environment we must start evaluation. A simplification could be to focus on closed terms only, since evaluation of any closed term can be started in the empty environment. The reduct of a closed term is necessarily, again, a closed term. This suggests that we can use the following translation from (closed) $\bar{\lambda}\mu$ terms to machine states.

**Definition 26.** Let $t \in \bar{\Lambda}\mu$ be a closed term. Then:

$$\{\!\{t\}\!\} = t\{\}$$

This, however, still does not work. Consider the following example: we have $\langle \lambda x.v_1 \parallel v_2 \cdot e\rangle \to_{wh} \langle v_1[v_2/x] \parallel e\rangle$ and we would like to have $\{\!\{\langle \lambda x.v_1 \parallel v_2 \cdot e\rangle\}\!\} \to^* \{\!\{\langle v_1[v_2/x] \parallel e\rangle\}\!\}$. But $\{\!\{\langle \lambda x.v_1 \parallel v_2 \cdot e\rangle\}\!\} = \langle (\lambda x.v_1)\{\} \parallel (v_2 \cdot e)\{\}\rangle \to \langle v_1\{x \mapsto v_2\{\}\} \parallel e\{\}\rangle$ and $\{\!\{\langle v_1[v_2/x] \parallel e\rangle\}\!\} = \langle (v_1[v_2/x])\{\} \parallel e\{\}\rangle$. Ideally, we would get $\langle v_1\{x \mapsto v_2\{\}\} \parallel e\{\}\rangle \to^* \langle (v_1[v_2/x])\{\} \parallel e\{\}\rangle$, which is not always the case.

For example, $\langle \lambda x.\lambda y.y \parallel v \cdot \alpha\rangle \to_{wh} \langle (\lambda y.y)[v/x] \parallel \alpha\rangle$, but $\{\!\{\langle \lambda x.\lambda y.y \parallel v \cdot \alpha\rangle\}\!\} = \langle (\lambda x.\lambda y.y)\{\} \parallel (v \cdot \alpha)\{\}\rangle \to \langle (\lambda y.y)\{x \mapsto v\{\}\} \parallel \alpha\{\}\rangle$ and $\{\!\{\langle (\lambda y.y)[v/x] \parallel \alpha\rangle\}\!\} = \langle ((\lambda y.y)[v/x])\{\} \parallel \alpha\{\}\rangle = \langle (\lambda y.y)\{\} \parallel \alpha\{\}\rangle$ and, hence, $\langle (\lambda y.y)\{x \mapsto v\{\}\} \parallel \alpha\{\}\rangle \nrightarrow \langle (\lambda y.y)\{\} \parallel \alpha\{\}\rangle$.

We can, however, prove that: for any closed, well-typed term $t \in \bar{\Lambda}\mu$, there exist environments $E_1$ and $E_2$, such that $\mu\alpha.\langle t\{\} \parallel \alpha\{\}\rangle \to^* \mu\alpha.\langle (\lambda x.v)\{E_1\} \parallel \alpha\{E_2\}\rangle$ and that, moreover, $\mu\alpha.\langle t \parallel \alpha\rangle \to^*_{wh} [\![\mu\alpha.\langle (\lambda x.v)\{E_1\} \parallel \alpha\{E_2\}\rangle]\!]$. I.e. that there exist environments $E_1$ and $E_2$, such that the following diagram commutes:

$$\begin{array}{ccc} \mu\alpha.\langle t \parallel \alpha\rangle & \xrightarrow{\quad\{\!\{-\}\!\}\quad} & \mu\alpha.\langle t\{\} \parallel \alpha\{\}\rangle \\ \Big\downarrow{\scriptstyle wh^*} & & \Big\downarrow{\scriptstyle *} \\ [\![\mu\alpha.\langle (\lambda x.v)\{E_1\} \parallel \alpha\{E_2\}\rangle]\!] & \xleftarrow[[\![-]\!]]{} & \mu\alpha.\langle (\lambda x.v)\{E_1\} \parallel \alpha\{E_2\}\rangle \end{array}$$

Such completeness lemma is proven by De Groote in [DG98] for his abstract machine that evaluates $\lambda\mu$-terms. We use the same proof technique here. Before we can prove completeness of the machine, we need an additional lemma.

**Definition 27.** The size $|E|$ of a machine environment $E$ is defined as follows:

$$|\{\}| = 0$$
$$|\{\nu \mapsto t\{E'\}, E\}| = |\{E\}| + |\{E'\}| + 1$$

**Lemma 33.** *The machine reductions are strongly normalizing for well-typed terms. That is: if the machine is loaded with a well-typed term, it cannot run forever.*

*Proof.* An infinite sequence of reductions cannot contain an infinite amount of (3) or (4) reductions, since, by Lemma 32, that would give rise to an infinite sequence of $\rightarrow_{\lambda\mu}$ contractions on the unloaded terms, which contradicts Lemma 14.

For reduction (1) we have $\langle x\{x \mapsto v\{E\}, E_1\} \parallel e\{E_2\}\rangle \rightarrow \langle v\{E\} \parallel e\{E_2\}\rangle$ and $|\{x \mapsto v\{E\}, E_1\}| + |\{E_2\}| = |\{E_1\}| + |\{E\}| + 1 + |\{E_2\}| > |\{E\}| + |\{E_e\}|$ and, hence, reduction (1) decreases the total size of the environments. The same holds for reduction (2) and, hence, we cannot have an infinite sequence of (1) or (2) reductions. □

**Corollary 4.** *The machine is confluent for well-typed terms: In Lemma 31 we have shown that the machine is locally confluent. In Lemma 33 we have shown that the reductions of the machine are strongly normalizing. Then, by Newman's lemma, we obtain confluence.*

We are now ready to prove completeness of the machine.

**Lemma 34.** *The machine is complete for the weak head reduction strategy defined in Definition 23. Let $t \in \bar{\Lambda}\mu$ be a closed, well-typed term. Then there exist environments $E_1$ and $E_2$, such that:*

$$\mu\alpha.\langle t\{\} \parallel \alpha\{\}\rangle \rightarrow^* \mu\alpha.\langle(\lambda x.v)\{E_1\} \parallel \alpha\{E_2\}\rangle \text{ , (for } \alpha \text{ fresh)}$$

*Moreover, $\mu\alpha.\langle t \parallel \alpha\rangle \rightarrow^*_{wh} [\![\mu\alpha.\langle(\lambda x.v)\{E_1\} \parallel \alpha\{E_2\}\rangle]\!]$.*

*Proof.* First note that if $t$ is a closed, well-typed term, then $t' = \mu\alpha.\langle t \parallel \alpha\rangle$ (for $\alpha$ fresh) is also a closed, well-typed term and, hence, we must show that there exist environments $E_1$ and $E_2$ such that $t'\{\} = \mu\alpha.\langle t\{\} \parallel \alpha\{\}\rangle \rightarrow^* \mu\alpha.\langle(\lambda x.v)\{E_1\} \parallel \alpha\{E_2\}\rangle$. Moreover, by Lemma 32, we must have that each intermediate machine state in the reduction sequence corresponds to a closed term. Now suppose that the machine does not reach a state $\mathcal{M} = \mu\alpha.\langle(\lambda x.v)\{E_1\} \parallel \alpha\{E_2\}\rangle$. Then either the machine must halt on another state or it must run forever.

*The machine cannot halt on another state.* We have the following states, other than $\mathcal{M}$, that the machine can halt on:

1. $\mu\alpha.\langle x\{E_1\} \parallel e\{E_2\}\rangle$ where $E_1(x)$ is undefined.
2. $\mu\alpha.\langle v\{E_1\} \parallel \beta\{E_2\}\rangle$ where $E_2(\beta)$ is undefined.

However, in state (1), $x$ corresponds to a free-variable and in (2), $\beta$ corresponds to a free variable. In both cases we get a contradiction with the fact that $t'$ is a closed term.

*The machine cannot run forever.* By Lemma 33.

So, indeed, the machine eventually reaches a state $\mu\alpha.\langle(\lambda x.v)\{E_1\} \parallel \alpha\{E_2\}\rangle$, for some environments $E_1$ and $E_2$. Then, by Lemma 32, we get $\mu\alpha.\langle t\{\} \parallel \alpha\{\}\rangle \rightarrow^* \mu\alpha.\langle(\lambda x.v)\{E_1\} \parallel \alpha\{E_2\}\rangle$, as required. □

**Example 15.** Consider the term $t = \mu\alpha.\langle \lambda x.\lambda y.x \parallel (\lambda z.z)\cdot\alpha\rangle$, which is a closed, well-typed term of type $B \to (A \to A)$. But then also $t' = \mu\beta.\langle \mu\alpha.\langle \lambda x.\lambda y.x \parallel (\lambda z.z)\cdot\alpha\rangle \parallel \beta\rangle$ is a closed, well-typed term (of the same type) and $\{t'\} = \mu\beta.\langle \mu\alpha.\langle (\lambda x.\lambda y.x)\{\} \parallel ((\lambda z.z)\cdot\alpha)\{\}\rangle \parallel \beta\{\}\rangle$. We get the following reduction of $t'$ using our machine:

$$
\begin{aligned}
&\mu\beta.\langle \mu\alpha.\langle (\lambda x.\lambda y.x)\{\} \parallel ((\lambda z.z)\cdot\alpha)\{\}\rangle \parallel \beta\{\}\rangle \\
&\to_4 \mu\beta.\langle (\lambda x.\lambda y.x)\{\alpha \mapsto \beta\{\}\} \parallel ((\lambda z.z)\cdot\alpha)\{\alpha \mapsto \beta\{\}\}\rangle \\
&\to_3 \mu\beta.\langle (\lambda y.x)\{x \mapsto (\lambda z.z)\{\alpha \mapsto \beta\{\}\}, \alpha \mapsto \beta\{\}\} \parallel \alpha\{\alpha \mapsto \beta\{\}\}\rangle \\
&\to_2 \mu\beta.\langle (\lambda y.x)\{x \mapsto (\lambda z.z)\{\alpha \mapsto \beta\{\}\}, \alpha \mapsto \beta\{\}\} \parallel \beta\{\}\rangle
\end{aligned}
$$

Now, for this last machine state we have:

$$
\begin{aligned}
&[\![\mu\beta.\langle (\lambda y.x)\{x \mapsto (\lambda z.z)\{\alpha \mapsto \beta\{\}\}, \alpha \mapsto \beta\{\}\} \parallel \beta\{\}\rangle]\!] \\
&= \mu\beta.\langle \textsc{subs}(\lambda y.x, \{x \mapsto (\lambda z.z)\{\alpha \mapsto \beta\{\}\}, \alpha \mapsto \beta\{\}\}) \parallel \textsc{subs}(\beta, \{\})\rangle \\
&= \mu\beta.\langle \textsc{subs}((\lambda y.x)[\textsc{subs}(\beta, \{\})/\alpha], \{x \mapsto (\lambda z.z)\{\alpha \mapsto \beta\{\}\}\}) \parallel \beta\rangle \\
&= \mu\beta.\langle \textsc{subs}(\lambda y.x, \{x \mapsto (\lambda z.z)\{\alpha \mapsto \beta\{\}\}\}) \parallel \beta\rangle \\
&= \mu\beta.\langle \textsc{subs}((\lambda y.x)[\textsc{subs}(\lambda z.z, \{\alpha \mapsto \beta\{\}\})/x], \{\}) \parallel \beta\rangle \\
&= \mu\beta.\langle \lambda y.\textsc{subs}(\lambda z.z, \{\alpha \mapsto \beta\{\}\}) \parallel \beta\rangle \\
&= \mu\beta.\langle \lambda y.(\lambda z.z)[\textsc{subs}(\beta, \{\})/\alpha] \parallel \beta\rangle \\
&= \mu\beta.\langle \lambda y.\lambda z.z \parallel \beta\rangle
\end{aligned}
$$

Moreover, using the weak head reduction strategy, we get:

$$
\begin{aligned}
t' &= \mu\beta.\langle \mu\alpha.\langle \lambda x.\lambda y.x \parallel (\lambda z.z)\cdot\alpha\rangle \parallel \beta\rangle \\
&\to_{wh} \mu\beta.\langle \lambda x.\lambda y.x \parallel (\lambda z.z)\cdot\beta\rangle \\
&\to_{wh} \mu\beta.\langle \lambda y.\lambda z.z \parallel \beta\rangle
\end{aligned}
$$

*end of example*

# Chapter 4

# The $\bar{\lambda}\mu\tilde{\mu}$-calculus

In this chapter we investigate some properties of the $\bar{\lambda}\mu\tilde{\mu}$-calculus, which was introduced by Curien and Herbelin in [CH00] (also see [Her05]). The $\bar{\lambda}\mu\tilde{\mu}$-calculus is an elegant combination of Herbelin's $\bar{\lambda}$-calculus [Her95] and Parigot's $\lambda\mu$-calculus [Par92]. Its sequent calculus foundation makes $\bar{\lambda}\mu\tilde{\mu}$ ideal to investigate some of the dualities of computation such as computation/context and call-by-name/call-by-value.

The $\bar{\lambda}\mu\tilde{\mu}$-calculus [CH00] is obtained by extending the $\bar{\lambda}\mu$-calculus with the new binding operator $\tilde{\mu}$ and an appropriate reduction rule. This operator can be interpreted as a **let** binding, which can be used, for example, to evaluate arguments to functions before evaluating the function itself. This allows call-by-name and call-by-value reduction strategies to live side by side in one calculus. The syntax is divided into the same three categories as before.

**Definition 28.** The set $\bar{\Lambda}\mu\tilde{\mu}$ of $\bar{\lambda}\mu\tilde{\mu}$ terms is defined by the following grammars.

$$\begin{array}{lrcl} \text{Commands} & c & ::= & \langle v \parallel e \rangle \\ \text{Terms} & v & ::= & x \mid \lambda x.v \mid \mu\alpha.c \\ \text{Contexts} & e & ::= & \beta \mid \tilde{\mu}x.c \mid v \cdot e \end{array}$$

We let $t$ range over arbitrary $\bar{\lambda}\mu\tilde{\mu}$ terms i.e. $t \in \bar{\Lambda}\mu\tilde{\mu}$. The new context construct $\tilde{\mu}x.c$ binds $x$ in $c$. The typing system uses the same three judgements as before, one for each syntactic category:

$$c : (\Gamma \vdash \Delta) \qquad \Gamma \vdash v : A \mid \Delta \qquad \Gamma \mid e : A \vdash \Delta$$

The typing rules are included in Figure 4. For details about the underlying logic we refer to [CH00]. On top of these typing rules, we can, of course, add the same weakening rules as for $\bar{\lambda}\mu$.

In Example 8 from Chapter 3 we already mentioned that, in a call-by-value strategy, we must first reduce arguments (to a function) to values before we can apply the function to those arguments. That is: in terms like $I^x(I^y I^z)$, we are forced to reduce the application $I^y I^z$ to the value $I^z$ before applying $I^x$. So we must be able to freeze the evaluation of $I^x$ and start reducing its arguments first.

The well known construct **let** $x = N$ **in** $M$ (see e.g. [HZ09]) can be used to achieve just that. Suppose we want to evaluate the application $I^x(I^y I^z)$.

$$\frac{}{\Gamma \mid \alpha : A \vdash \alpha : A, \Delta}\ \mathrm{AX}_l \qquad \frac{}{\Gamma, x : A \vdash x : A \mid \Delta}\ \mathrm{AX}_r$$

$$\frac{\Gamma \vdash v : A \mid \Delta \quad \Gamma \mid e : B \vdash \Delta}{\Gamma \mid (v \cdot e) : A \to B \vdash \Delta}\ \to_l \qquad \frac{\Gamma, x : A \vdash v : B \mid \Delta}{\Gamma \vdash \lambda x.v : A \to B \mid \Delta}\ \to_r$$

$$\frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha.c : A \mid \Delta}\ \mu \qquad \frac{c : (\Gamma, x : A \vdash \Delta)}{\Gamma \mid \tilde{\mu}x.c : A \vdash \Delta}\ \tilde{\mu}$$

$$\frac{\Gamma \vdash v : A \mid \Delta \quad \Gamma \mid e : A \vdash \Delta}{\langle v \parallel e \rangle : (\Gamma \vdash \Delta)}\ \mathrm{CM}$$

Figure 4.1: Typing rules for $\bar{\lambda}\mu\tilde{\mu}$

Then we could start by writing it as **let** $k = I^y I^z$ **in** $I^x k$, where we intend to reduce $I^y I^z$ before passing it to $I^x k$. In terms of contexts: if $I^x(I^y I^z)$ is evaluated in context $E$, then actually we want to evaluate $K[I^y I^z]$ where $K = E[\textbf{let } k = [\,] \textbf{ in } I^x k]$. This is exactly what the new operator $\tilde{\mu}$ allows us to express: we write $\tilde{\mu}x.\langle v \parallel e\rangle$ for $e[\textbf{let } x = [\,] \textbf{ in } v]$. A context $v_1 \cdot \ldots \cdot v_k \cdot \tilde{\mu}x.\langle v \parallel e\rangle$ can then be read as $e[\textbf{let } x = [[\,]v_1 \ldots v_k] \textbf{ in } v]$. So, a $\tilde{\mu}$-abstraction can be seen as a function, waiting for an argument.

**Definition 29.** We have the following reductions in $\bar{\lambda}\mu\tilde{\mu}$:

$$\begin{array}{lrcl} (\lambda') & \langle \lambda x.v_1 \parallel v_2 \cdot e\rangle & \to & \langle v_2 \parallel \tilde{\mu}x.\langle v_1 \parallel e\rangle\rangle \\ (\mu) & \langle \mu\alpha.c \parallel e\rangle & \to & c[e/\alpha] \\ (\tilde{\mu}) & \langle v \parallel \tilde{\mu}x.c\rangle & \to & c[v/x] \end{array}$$

Where substitution is capture-free with respect to both kinds of variables.

We adopt the convention that all names of bound variables are chosen maximally fresh and different from unbound variables. Therefore we get that $\tilde{\mu}$ only binds free occurrences of $x$ in $v_1$ in the right hand side of the $\lambda'$ reduction.

Note that the $\lambda'$ reduction actually differs from the $\lambda$ reduction we had for $\bar{\lambda}\mu$ in Chapter 3. The intuition is that we can now freeze the evaluation of a function $\lambda x.v$, by encoding it as the context **let** $x = [\,]$ **in** $v$, and evaluate an argument to a value first. The set of values is defined as follows:

**Definition 30.** The set of $\bar{\lambda}\mu\tilde{\mu}$ values is generated by the following grammar:

$$V ::= x \ \mid \ \lambda x.v$$

We let $V_1, V_2, \ldots$ range over values, not to mistake with $v_1, v_2, \ldots$, which range over $\bar{\lambda}\mu\tilde{\mu}$ terms.

Let us check that the reductions in $\bar{\lambda}\mu\tilde{\mu}$ satisfy subject reduction. As before, we need the following substitution lemma.

**Lemma 35.** *Typing is preserved under substitution. That is: if*

*(A)* $\Gamma \mid e' : A' \vdash \Delta$,

*(B)* $\Gamma \vdash v : A \mid \alpha : A', \Delta$ *and*

*(C)* $\Gamma \mid e : A \vdash \alpha : A', \Delta$

*then the following statements hold:*

*(1)* $\Gamma \mid e[e'/\alpha] : A \vdash \Delta$

*(2)* $\Gamma \vdash v[e'/\alpha] : A \mid \Delta$

*Proof.* Simultaneously, by induction on the structure of $e, v$. We only need to check one case, since the other cases are already checked in Lemma 12.

- (1) If $e = \tilde{\mu}x.\langle v' \parallel e''\rangle$, then we get $\langle v' \parallel e''\rangle : (\Gamma, x : A \vdash \alpha : A', \Delta)$ by (C) and the $\tilde{\mu}$ typing rule. Then, by the CM typing rule, we have $\Gamma, x : A \vdash v' : B \mid \alpha : A', \Delta$ and $\Gamma, x : A \mid e'' : B \vdash \alpha : A', \Delta$ for some $B$. By applying the induction hypothesis to these two judgements, we get both $\Gamma, x : A \vdash v'[e'/\alpha] : B \mid \Delta$ and $\Gamma, x : A \mid e''[e'/\alpha] : B \vdash \Delta$. Hence, we have the following derivation, as required ($e[e'/\alpha] = (\tilde{\mu}x.\langle v' \parallel e''\rangle)[e'/\alpha] = \tilde{\mu}x.\langle v'[e'/\alpha] \parallel e''[e'/\alpha]\rangle$):

$$\dfrac{\dfrac{\Gamma, x : A \vdash v'[e'/\alpha] : B \mid \Delta \quad \Gamma, x : A \mid e''[e'/\alpha] : B \vdash \Delta}{\langle v'[e'/\alpha] \parallel e''[e'/\alpha]\rangle : (\Gamma, x : A \vdash \Delta)}\ \text{CM}}{\Gamma \mid \tilde{\mu}x.\langle v'[e'/\alpha] \parallel e''[e'/\alpha]\rangle : A \vdash \Delta}\ \tilde{\mu}$$

□

**Remark 4.** Note that the contents of Corollary 1 and Remark 2 also hold for Lemma 35.

**Lemma 36.** *The reductions in* $\bar{\lambda}\mu$ *satisfy subject reduction. That is: if* $\langle v \parallel e\rangle : (\Gamma \vdash \Delta)$ *and* $\langle v \parallel e\rangle \to \langle v' \parallel e'\rangle$*, then* $\langle v' \parallel e'\rangle : (\Gamma \vdash \Delta)$.

*Proof.* We only have to check the $\to_{\lambda'}$ and the $\to_{\tilde{\mu}}$ reductions, since the $\to_{\mu}$ reduction was already checked in Lemma 13.

- Suppose $\langle v \parallel e\rangle = \langle \lambda x.v_1 \parallel v_2 \cdot e\rangle$, then $\langle v \parallel e\rangle \to_{\lambda'} \langle v_2 \parallel \tilde{\mu}x.\langle v_1 \parallel e\rangle\rangle$. We have:

$$\dfrac{\dfrac{\Gamma, x : A \vdash v_1 : B \mid \Delta}{\Gamma \vdash \lambda x.v_1 : A \to B \mid \Delta}\ \to_r \quad \dfrac{\Gamma \vdash v_2 : A \mid \Delta \quad \Gamma \mid e : B \vdash \Delta}{\Gamma \mid v_2 \cdot e : A \to B \vdash \Delta}\ \to_l}{\langle \lambda x.v_1 \parallel v_2 \cdot e\rangle : (\Gamma \vdash \Delta)}\ \text{CM}$$

$$\longrightarrow_{\lambda'}$$

$$\dfrac{\Gamma \vdash v_2 : A \mid \Delta \qquad \dfrac{\dfrac{\Gamma, x : A \vdash v_1 : B \mid \Delta \quad \Gamma, x : A \mid e : B \vdash \Delta}{\langle v_1 \parallel e\rangle : (\Gamma, x : A \vdash \Delta)}\ \text{cm}}{\Gamma \mid \tilde{\mu}x.\langle v_1 \parallel e\rangle : A \vdash \Delta}\ \tilde{\mu}}{\langle v_2 \parallel \tilde{\mu}x.\langle v_1 \parallel e\rangle\rangle : (\Gamma \vdash \Delta)}\ \text{CM}$$

- Suppose $\langle v \parallel e \rangle = \langle v \parallel \tilde{\mu}x.c \rangle$, then $\langle v \parallel e \rangle \to_{\tilde{\mu}} c[v/x]$. We have:

$$\dfrac{\Gamma \vdash v : A \mid \Delta \quad \dfrac{c : (\Gamma, x : A \vdash \Delta)}{\Gamma \vdash \tilde{\mu}x.c : A \mid \Delta}\ \tilde{\mu}}{\langle v \parallel \tilde{\mu}x.c \rangle : (\Gamma \vdash \Delta)}\ \text{CM}$$

  Using Remark 4 we get $c[v/x] : (\Gamma \vdash \Delta)$ as desired.

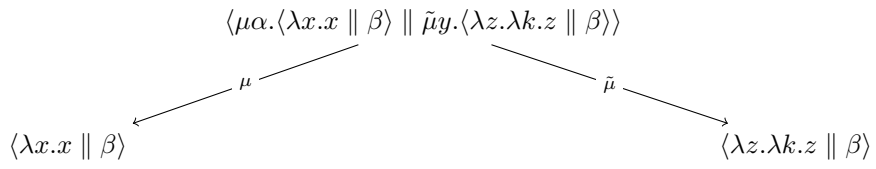
□

**Lemma 37.** *The reductions in $\bar{\lambda}\mu\tilde{\mu}$ are strongly normalizing. That is: the process of reduction always terminates.*

*Proof.* See [Her05]. □

Observe that the reductions in $\bar{\lambda}\mu\tilde{\mu}$ are not confluent: for $t = \langle \mu\alpha.c_1 \parallel \tilde{\mu}x.c_2 \rangle$ we get $t \to_{\mu} c_1[\tilde{\mu}x.c_2/\alpha]$ and $t \to_{\tilde{\mu}} c_2[\mu\alpha.c_1/x]$. But the critical pair $(c_1[\tilde{\mu}x.c_2/\alpha], c_2[\mu\alpha.c_1/x])$ does not always converge.

**Example 16.** Consider, for example, the specific case where $t = \langle \mu\alpha.\langle \lambda x.x \parallel \beta \rangle \parallel \tilde{\mu}y.\langle \lambda z.\lambda k.z \parallel \beta \rangle \rangle$, which is a reduct of the command $\langle \lambda y.\lambda z.\lambda k.z \parallel \mu\alpha.\langle \lambda x.x \parallel \beta \rangle \cdot \beta \rangle$. We get the following reduction graph, where $\langle \lambda y.y \parallel \beta \rangle$ and $\langle \lambda z.\lambda k.z \parallel \beta \rangle$ do not converge:




*end of example*

We have already mentioned, in the introduction, that the call-by-name and call-by-value reduction strategies live side by side in $\bar{\lambda}\mu\tilde{\mu}$. Picking one of the reduction strategies amounts to giving priority to either the $\mu$ reduction, for call-by-value, or to the $\tilde{\mu}$ reduction, for call-by-name, in case one encounters the "critical command" $\langle \mu\alpha.c_1 \parallel \tilde{\mu}x.c_2 \rangle$.

But why is that the case? First observe that, if we give priority to the $\tilde{\mu}$ reduction, we may adapt the $\lambda'$ reduction so that it immediately contracts the created $\tilde{\mu}$ redex. We get: $\langle \lambda x.v_1 \parallel v_2 \cdot e \rangle \to_{\lambda'} \langle v_2 \parallel \tilde{\mu}x.\langle v_1 \parallel e \rangle \rangle \to_{\tilde{\mu}} \langle v_1 \parallel e \rangle[v_2/x] = \langle v_1[v_2/x] \parallel e \rangle$, since $\tilde{\mu}$ only binds free occurrences of $x$ in $v_1$. So, in fact, we retrieve the $\lambda$ reduction from $\bar{\lambda}\mu$ in this way, which is a call-by-name reduction (see Chapter 3). I.e. the $\tilde{\mu}$ redexes, created by the reduction system itself can be avoided. Furthermore we can perform the $\tilde{\mu}$ reduction on any command of the form $\langle v \parallel \tilde{\mu}x.c \rangle$, independent of the form of $v$. In particular $v$ can be a $\mu$-abstraction, which is used to encode applications. But then the $\tilde{\mu}$ reduction amounts to performing a call-by-name reduction.

If we give priority to the $\mu$ reduction, then any command of the form $\langle v \parallel e \rangle$ can be reduced (by performing $\mu$-reduction as many times as possible) to a command $\langle V \parallel e' \rangle$ where $V$ is not a $\mu$-abstraction i.e. $V$ is a value in the sense

of Definition 30. Once we have obtained a command $\langle V \parallel e' \rangle$, we may perform a $\tilde{\mu}$ reduction whenever $e' = \tilde{\mu}x.c$ and, hence, we will substitute a value for $x$ in $c$. So, indeed, by giving priority to the $\mu$ reduction, we obtain a call-by-value strategy.

**Example 17.** Consider the following call-by-value reduction of $\mu\alpha.\langle I^x \parallel \mu\beta.\langle I^y \parallel I^z \cdot \beta\rangle \cdot \alpha\rangle$, which is the $\bar{\lambda}\mu\tilde{\mu}$ equivalent of $I^x(I^yI^z)$:

$$\begin{aligned}
\mu\alpha.\langle I^x \parallel \mu\beta.\langle I^y \parallel I^z \cdot \beta\rangle \cdot \alpha\rangle &\to_{\lambda'} \mu\alpha.\langle \mu\beta.\langle I^y \parallel I^z \cdot \beta\rangle \parallel \tilde{\mu}x.\langle x \parallel \alpha\rangle\rangle \\
&\to_{\mu} \mu\alpha.\langle I^y \parallel I^z \cdot \tilde{\mu}x.\langle x \parallel \alpha\rangle\rangle \\
&\to_{\lambda'} \mu\alpha.\langle I^z \parallel \tilde{\mu}y.\langle y \parallel \tilde{\mu}x.\langle x \parallel \alpha\rangle\rangle\rangle \\
&\to_{\tilde{\mu}} \mu\alpha.\langle I^z \parallel \tilde{\mu}x.\langle x \parallel \alpha\rangle\rangle \\
&\to_{\tilde{\mu}} \mu\alpha.\langle I^z \parallel \alpha\rangle
\end{aligned}$$

And its call-by-name counterpart:

$$\begin{aligned}
\mu\alpha.\langle I^x \parallel \mu\beta.\langle I^y \parallel I^z \cdot \beta\rangle \cdot \alpha\rangle &\to_{\lambda'} \mu\alpha.\langle \mu\beta.\langle I^y \parallel I^z \cdot \beta\rangle \parallel \tilde{\mu}x.\langle x \parallel \alpha\rangle\rangle \\
&\to_{\tilde{\mu}} \mu\alpha.\langle \mu\beta.\langle I^y \parallel I^z \cdot \beta\rangle \parallel \alpha\rangle \\
&\to_{\mu} \mu\alpha.\langle I^y \parallel I^z \cdot \alpha\rangle \\
&\to_{\lambda'} \mu\alpha.\langle I^z \parallel \tilde{\mu}y.\langle y \parallel \alpha\rangle\rangle \\
&\to_{\tilde{\mu}} \mu\alpha.\langle I^z \parallel \alpha\rangle
\end{aligned}$$

*end of example*

Let us make this "giving priority to $\mu$ or $\tilde{\mu}$-redexes" more precise. We can adapt the reduction system to automatically give priority to either $\mu$ or $\tilde{\mu}$-redexes. For instance, we could use our definition of values (Definition 30) and change our $\tilde{\mu}$-reduction into:

$$\langle V \parallel \tilde{\mu}x.c\rangle \to_{\tilde{\mu}'} c[V/x]$$

If we leave the other reductions unchanged, then we have given priority to $\mu$-redexes and thereby obtain the call-by-value reduction strategy, since we can only use the $\tilde{\mu}'$ reduction if the left side of the command is *not* a $\mu$-abstraction i.e. if it is a value.

The other way around we can adapt the reduction system to automatically give priority to $\tilde{\mu}$-redexes.

**Definition 31.** The set of restricted contexts is defined by the grammar $E$:

$$E ::= \alpha \mid v \cdot e$$

Using the above definition we could change our $\mu$-reduction into:

$$\langle \mu\alpha.c \parallel E\rangle \to_{\mu'} c[E/\alpha]$$

If we leave the other reductions unchanged, we have given priority to $\tilde{\mu}$-redexes and thereby obtain the call-by-name reduction strategy, since we can only use the $\mu'$ reduction if the right side of the command is *not* a $\tilde{\mu}$-abstraction.

In both cases, the adaptation of the reduction system leads to a reduction system that is confluent for well-typed terms, as the following to lemmas show.

**Lemma 38.** *If we equip the reduction system of $\bar{\lambda}\mu\tilde{\mu}$ with the $\mu'$ ($\tilde{\mu}'$) rule instead of the $\mu$ ($\tilde{\mu}$) rule, then we obtain a reduction system that is strongly normalizing.*

*Proof.* By Lemma 37, the reductions in $\bar{\lambda}\mu\tilde{\mu}$ itself are strongly normalizing. In particular the $\rightarrow_\mu$ and $\rightarrow_{\tilde{\mu}}$ are strongly normalizing. But, in a trivial way, we have $\rightarrow_{\mu'} \subset \rightarrow_\mu$ and $\rightarrow_{\tilde{\mu}'} \subset \rightarrow_{\tilde{\mu}}$. Hence, the $\mu'$ and $\tilde{\mu}'$ reductions must be strongly normalizing. □

**Lemma 39.** *If we equip the reduction system of $\bar{\lambda}\mu\tilde{\mu}$ with the $\mu'$ ($\tilde{\mu}'$) rule instead of the $\mu$ ($\tilde{\mu}$) rule, then we obtain a reduction system that is confluent for well-typed terms.*

*Proof.* In either case, there are no more critical pairs. Then, by Lemma 38 and Newman's Lemma, we obtain confluence. □

Example 17 gives an example of how these reductions work. In the call-by-value case we must replace all $\tilde{\mu}$ reductions with $\tilde{\mu}'$ and in the call-by-name case we must replace all $\mu$ reductions with $\mu'$.

## 4.1 Exception handling

Recall that, in $\bar{\lambda}\mu$, we were able to define an exception handling mechanism with the operators `catch` and `throw` . For these operators we had the following reductions:

1. $\mu\alpha.\langle \mathtt{throw}_\beta\ v \parallel e\rangle \rightarrow \mathtt{throw}_\beta\ v$, provided that $\alpha \notin FV(v)$
2. $\mathtt{catch}_\alpha\ (\mathtt{throw}_\alpha\ v) \rightarrow \mathtt{catch}_\alpha\ v$
3. $\mathtt{catch}_\alpha\ (\mathtt{throw}_\beta\ v) \rightarrow \mathtt{throw}_\beta\ v$, provided that $\alpha \notin FV(v)$
4. $\mathtt{throw}_\beta\ (\mathtt{throw}_\alpha\ v) \rightarrow \mathtt{throw}_\alpha\ v$

Note that we have the same reductions for `catch` and `throw` in $\bar{\lambda}\mu\tilde{\mu}$, since these reductions were all defined in terms of $\bar{\lambda}\mu$'s reduction system, which is a subset of $\bar{\lambda}\mu\tilde{\mu}$'s reduction system. In $\bar{\lambda}\mu\tilde{\mu}$, however, we have the option to perform call-by-value reduction steps due to the modified $\lambda$ reduction. This allows us to add the following reduction for `catch` and `throw` :

**Lemma 40.** *We have the following additional reduction for `catch` and `throw` in $\bar{\lambda}\mu\tilde{\mu}$:*

$$\mu\alpha.\langle \lambda x.v_1 \parallel (\mathtt{throw}_\beta\ v_2)\cdot e\rangle \rightarrow \mathtt{throw}_\beta\ v_2 \quad \textit{, provided that } \alpha \notin FV(v_2)$$

*Proof.* We have: $\mu\alpha.\langle \lambda x.v_1 \parallel (\mathtt{throw}_\beta\ v_2)\cdot e\rangle = \mu\alpha.\langle \lambda x.v_1 \parallel (\mu\gamma.\langle v_2 \parallel \beta\rangle)\cdot e\rangle \rightarrow_{\lambda'} \mu\alpha.\langle \mu\gamma.\langle v_2 \parallel \beta\rangle \parallel \tilde{\mu}x.\langle v_1 \parallel e\rangle\rangle \rightarrow_\mu \mu\alpha.\langle v_2 \parallel \beta\rangle = \mathtt{throw}_\beta\ v_2$ □

Indeed, we must have that $\alpha \notin FV(v_2)$, as the following example shows:

**Example 18.** Consider the term $t = \mu\alpha.\langle \lambda x.v_1 \parallel (\mathtt{throw}_\beta\ v_2) \cdot e\rangle$ with $v_2 = \mu\delta.\langle y \parallel \alpha\rangle$, then (using our new reduction):

$$\begin{aligned}
&\mu\alpha.\langle \lambda x.v_1 \parallel (\mathtt{throw}_\beta\ (\mu\delta.\langle y \parallel \alpha\rangle)) \cdot e\rangle \\
&\to \mu\alpha.\langle \mu\delta.\langle y \parallel \alpha\rangle \parallel \beta\rangle \\
&\neq \mathtt{throw}_\beta\ (\mu\delta.\langle y \parallel \alpha\rangle) \quad \text{, since } \alpha \in FV(\mu\delta.\langle y \parallel \alpha\rangle)
\end{aligned}$$

*end of example*

## 4.2 An environment machine for $\bar{\lambda}\mu\tilde{\mu}$

In Section 2.2.2 we already mentioned that the machine we introduced is a restriction of Curien and Herbelin's machine for $\bar{\lambda}\mu\tilde{\mu}$. In this section we first extend the notion of weak head reduction to $\bar{\lambda}\mu\tilde{\mu}$ and then introduce the full $\bar{\lambda}\mu\tilde{\mu}$ machine. Thereafter, we will extend our proofs of correctness and completeness to this extended machine.

**Definition 32.** The weak head reduction strategy for $\bar{\lambda}\mu\tilde{\mu}$ is defined by the following rules:

1. $\langle \lambda x.v \parallel v' \cdot e\rangle \to_{wh} \langle v' \parallel \tilde{\mu}x.\langle v \parallel e\rangle\rangle$
2. $\langle \mu\alpha.c \parallel e\rangle \to_{wh} c[e/\alpha]$
3. $\langle v \parallel \tilde{\mu}x.c\rangle \to_{wh} c[v/x]$
4. $t \to_{wh}^* t$
5. $\dfrac{c \to_{wh}^* c'}{\mu\alpha.c \to_{wh}^* \mu\alpha.c'}$

Indeed, the above definition of weak head reduction is non-deterministic and non-confluent (see Example 16). However, using our earlier results of this chapter, we can always make the system confluent (for well-typed terms) by giving priority to either $\mu$-redexes (which yields a call-by-value evaluation strategy) or to $\tilde{\mu}$-redexes (which yields a call-by-name evaluation strategy). We can now define a specific call-by-name and call-by-value version of weak head reduction.

**Definition 33.** The call-by-name weak head reduction strategy for $\bar{\lambda}\mu\tilde{\mu}$ is obtained by replacing reduction (2) with

$$\langle \mu\alpha.c \parallel E\rangle \to_{wh} c[E/\alpha] \quad \text{, with } E \text{ as in Definition 31}$$

**Definition 34.** The call-by-value weak head reduction strategy for $\bar{\lambda}\mu\tilde{\mu}$ is obtained by replacing reduction (3) with

$$\langle V \parallel \tilde{\mu}x.c\rangle \to_{wh} c[V/x] \quad \text{, with } V \text{ as in Definition 30}$$

However, for proving the correctness and completeness properties of a machine, we do not have to restrict ourselves to one of these deterministic strategies. We can just prove those properties for the non-deterministic strategy of Definition 32 and then restrict them to one of the deterministic cases later on.

**Definition 35.** The environment machine for $\bar{\lambda}\mu\tilde{\mu}$ ([CH00]) is defined by the following reductions:

1. $\langle x\{E_1\} \parallel e\{E_2\}\rangle \to \langle E_1(x) \parallel e\{E_2\}\rangle$ (if $E_1(x)$ is defined)
2. $\langle v\{E_1\} \parallel \alpha\{E_2\}\rangle \to \langle v\{E_1\} \parallel E_2(\alpha)\rangle$ (if $E_2(\alpha)$ is defined)
3. $\langle(\lambda x.v_1)\{E_1\} \parallel (v_2 \cdot e)\{E_2\}\rangle \to \langle v_2\{E_2\} \parallel \tilde{\mu}x.\langle v_1\{E_1\} \parallel e\{E_2\}\rangle\rangle$
4. $\langle\mu\alpha.\langle v\{E_1\} \parallel e_2\{E_2\}\rangle \parallel e_1\{E_3\}\rangle \to \langle v\{\alpha \mapsto e_1\{E_3\}, E_1\} \parallel e_2\{\alpha \mapsto e_1\{E_3\}, E_2\}\rangle$
5. $\langle v_1\{E_1\} \parallel \tilde{\mu}x.\langle v_2\{E_2\} \parallel e\{E_3\}\rangle\rangle \to \langle v_2\{x \mapsto v_1\{E_1\}, E_2\} \parallel e\{x \mapsto v_1\{E_1\}, E_3\}\rangle$

Note that reduction (3) actually differs from its $\bar{\lambda}\mu$ analogue and the one presented in [CH00]: instead of putting $v_2$ in the environment $E_l$ (i.e. substituting $v_2$ for $x$ in $v_1$) we give the machine a chance to perform call-by-value reduction by rewriting $\lambda x.v_1$ to a $\tilde{\mu}$-abstraction (a blocked computation). At the top-level we may apply the following equations:

$$\langle(\mu\alpha.\langle v \parallel e\rangle)\{E_1\} \parallel e'\{E_2\}\rangle = \langle\mu\alpha.\langle v\{E_1\} \parallel e\{E_1\}\rangle \parallel e'\{E_2\}\rangle$$
$$\langle v'\{E_1\} \parallel (\tilde{\mu}x.\langle v \parallel e\rangle)\{E_2\}\rangle = \langle v'\{E_1\} \parallel \tilde{\mu}x.\langle v\{E_2\} \parallel e\{E_2\}\rangle\rangle$$

**Lemma 41.** *The machine's reductions are locally confluent for all machine states but $\langle\mu\alpha.\langle v_1\{E_1\} \parallel e_1\{E_2\}\rangle \parallel \tilde{\mu}x.\langle v_2\{E_3\} \parallel e_2\{E_4\}\rangle\rangle$. That is: if $\mathcal{M}$ is a machine state and $\mathcal{M} \to \mathcal{M}_1$ and $\mathcal{M} \to \mathcal{M}_1$, then there exists a state $\mathcal{M}'$ such that $\mathcal{M}_1 \to^* \mathcal{M}'$ and $\mathcal{M}_2 \to^* \mathcal{M}'$.*

*Proof.* We only have to check the case where $\mathcal{M} = \langle x\{E_1\} \parallel \tilde{\mu}y.\langle v\{E_2\} \parallel e\{E_3\}\rangle\rangle$, since all other cases are handled in Lemma 31. So suppose $\mathcal{M} = \langle x\{E_1\} \parallel \tilde{\mu}y.\langle v\{E_2\} \parallel e\{E_3\}\rangle\rangle$ and $E_1(x)$ is defined, then $\mathcal{M} \to_1 \langle E_1(x) \parallel \tilde{\mu}y.\langle v\{E_2\} \parallel e\{E_3\}\rangle\rangle = \mathcal{M}_1$ and $\mathcal{M} \to_5 \langle v\{y \mapsto x\{E_1\}, E_2\} \parallel e\{y \mapsto x\{E_1\}, E_3\}\rangle = \mathcal{M}_2$. Note that $y \mapsto x\{E_1\} = y \mapsto E_1(x)$, since $E_1(x)$ is defined. Now if we take $\mathcal{M}' = \langle v\{y \mapsto x\{E_1\}, E_2\} \parallel e\{y \mapsto x\{E_1\}, E_3\}\rangle$, then $\mathcal{M}_1 \to_5 \mathcal{M}'$ and $\mathcal{M}_2 = \mathcal{M}'$, as required. □

We will now show that the machine is correct with respect to the notion of weak head reduction in Definition 32. The translation $[\![-]\!]$ from machine states to $\bar{\lambda}\mu\tilde{\mu}$ terms is the same as the one defined in 25.

**Lemma 42.** *The machine is correct for the weak head reduction system defined in Definition 32. Let $\mathcal{M}$ be a machine state. If $\mathcal{M} \to_{1,2} \mathcal{M}'$, then $[\![\mathcal{M}]\!] = [\![\mathcal{M}']\!]$. If $\mathcal{M} \to_{3,4,5} \mathcal{M}'$, then $[\![\mathcal{M}]\!] \to_{wh} [\![\mathcal{M}']\!]$.*

*Proof.* We only need to check the cases where $\mathcal{M} \to_3 \mathcal{M}'$ and $\mathcal{M} \to_5 \mathcal{M}'$, since the other cases are covered in Lemma 32.

- Suppose $\mathcal{M} = \langle(\lambda x.v_1)\{E_1\} \parallel (v_2 \cdot e)\{E_2\}\rangle$, then $\mathcal{M}' = \langle v_2\{E_2\} \parallel \tilde{\mu}x.\langle v_1\{E_1\} \parallel e\{E_2\}\rangle\rangle$ and:

$$\begin{aligned}
[\![\mathcal{M}]\!] &= [\![\langle(\lambda x.v_1)\{E_1\} \parallel (v_2 \cdot e)\{E_2\}\rangle]\!] \\
&= \langle \textsc{subs}(\lambda x.v_1, E_1) \parallel \textsc{subs}(v_2 \cdot e, E_2)\rangle \\
&= \langle \lambda x.\textsc{subs}(v_1, E_1) \parallel \textsc{subs}(v_2, E_2) \cdot \textsc{subs}(e, E_2)\rangle \\
&\to_{wh} \langle \textsc{subs}(v_2, E_2) \parallel \tilde{\mu}x.\langle \textsc{subs}(v_1, E_1) \parallel \textsc{subs}(e, E_2)\rangle\rangle \\
&= [\![\langle v_2\{E_2\} \parallel \tilde{\mu}x.\langle v_1\{E_1\} \parallel e\{E_2\}\rangle\rangle]\!] = [\![\mathcal{M}']\!]
\end{aligned}$$

- Suppose $\mathcal{M} = \langle v_1\{E_1\} \parallel \tilde{\mu}x.\langle v_2\{E_2\} \parallel e\{E_3\}\rangle\rangle$, then $\mathcal{M}' = \langle v_2\{x \mapsto v_1\{E_1\}, E_2\} \parallel e\{x \mapsto v_1\{E_1\}, E_3\}\rangle$ and:

$$\begin{aligned}
[\![\mathcal{M}]\!] &= [\![\langle v_1\{E_1\} \parallel \tilde{\mu}x.\langle v_2\{E_2\} \parallel e\{E_3\}\rangle\rangle]\!] \\
&= \langle \textsc{subs}(v_1, E_1) \parallel \tilde{\mu}x.\langle \textsc{subs}(v_2, E_2) \parallel \textsc{subs}(e, E_3)\rangle\rangle \\
&\to_{wh} \langle (\textsc{subs}(v_2, E_2))[\textsc{subs}(v_1, E_1)/x] \parallel \textsc{subs}(e, E_3)[\textsc{subs}(v_1, E_1)/x]\rangle \\
&= \langle \textsc{subs}(\textsc{subs}(v_2, E_2), \{x \mapsto v_1\{E_1\}\}) \parallel \textsc{subs}(\textsc{subs}(e, E_3), \{x \mapsto v_1\{E_1\}\})\rangle \\
&= \langle \textsc{subs}(v_2, \{\{x \mapsto v_1\{E_1\}\}, E_2\}) \parallel \textsc{subs}(e, \{\{x \mapsto v_1\{E_1\}\}, E_3\})\rangle \\
&= [\![\langle v_2\{\{x \mapsto v_1\{E_1\}\}, E_2\} \parallel e\{\{x \mapsto v_1\{E_1\}\}, E_3\}\rangle]\!] = [\![\mathcal{M}']\!]
\end{aligned}$$

□

Before we can prove that the machine is also complete for the notion of weak head reduction in $\bar{\lambda}\mu\tilde{\mu}$, we need an additional lemma.

**Lemma 43.** *The machine reductions are strongly normalizing for well-typed terms. That is: there are no infinite reduction paths.*

*Proof.* It remains to show that reductions (3) and (5) are strongly normalizing, since the other cases are already covered in Lemma 33. An infinite sequence of reductions cannot contain an infinite amount of (3) or (5) reductions, since, by Lemma 42, that would give rise to an infinite sequence $\to_{\lambda'\tilde{\mu}}$ contractions on the unloaded terms, which contradicts Lemma 37. □

**Lemma 44.** *The machine is complete for the weak head reduction system defined in Definition 32. Let $t \in \bar{\Lambda}\mu\tilde{\mu}$ be a closed, well-typed term. Then there exist environments $E_1$ and $E_2$, such that:*

$$\mu\alpha.\langle t\{\} \parallel \alpha\{\}\rangle \to^* \mu\alpha.\langle(\lambda x.v)\{E_1\} \parallel \alpha\{E_2\}\rangle \text{ , (for } \alpha \text{ fresh)}$$

*Moreover, $\mu\alpha.\langle t \parallel \alpha\rangle \to^*_{wh} [\![\mu\alpha.\langle(\lambda x.v)\{E_1\} \parallel \alpha\{E_2\}\rangle]\!]$.*

*Proof.* The same as for Lemma 34, but now using Lemma 43 to show that the machine cannot run forever. □

We can now restrict the machine to obtain the call-by-value and call-by-name specific versions. This is done in the same way as we restricted the reduction system of $\bar{\lambda}\mu\tilde{\mu}$ and the notion of weak head reduction before.

**Definition 36.** The call-by-name machine for $\bar{\lambda}\mu\tilde{\mu}$ is obtained by replacing reduction (4) with

$$\langle\mu\alpha.\langle v\{E_1\} \parallel e\{E_2\}\rangle \parallel E\{E_3\}\rangle \to_{(4)'} \langle v\{\alpha \mapsto E\{E_3\}, E_1\} \parallel e\{\alpha \mapsto E\{E_3\}, E_2\}\rangle$$

**Definition 37.** The call-by-value machine for $\bar{\lambda}\mu\tilde{\mu}$ is obtained by replacing reduction (5) with

$$\langle V\{E_1\} \parallel \tilde{\mu}x.\langle v\{E_2\} \parallel e\{E_3\}\rangle\rangle \to_{(5)'} \langle v\{x \mapsto V\{E_1\}, E_2\} \parallel e\{x \mapsto V\{E_1\}, E_3\}\rangle$$

**Lemma 45.** *The restricted machine from Definition 36 (Definition 37) satisfies the following properties:*

1. *The machine reductions are locally confluent,*
2. *The machine reductions are strongly normalizing,*
3. *The machine reductions are confluent for well-typed terms,*
4. *The machine is correct and complete for the weak head reduction strategy from Definition 33 (Definition 34).*

*Proof.*

1. By restricting machine reduction (4) (or (5) in the case of call-by-value) we no longer have that $\langle \mu\alpha.\langle v_1\{E_1\} \parallel e_1\{E_2\}\rangle \parallel \tilde{\mu}x.\langle v_2\{E_3\} \parallel e_2\{E_4\}\rangle\rangle$ is a "critical" machine state. Then, by Lemma 41, we obtain local confluence for both machines.

2. We already showed in Lemma 38, that the reductions of the original machine are strongly normalizing. In particular reductions (4) and (5) are strongly normalizing. But, trivially, $\to_{(4)'} \subset \to_{(4)}$ and $\to_{(5)'} \subset \to_{(5)}$ and, hence, we must have that both these reductions are strongly normalizing.

3. By the previous two items and Newman's Lemma we obtain confluence for the machines with restricted reductions.

4. We will only describe the changes in the existing lemmas. The correctness lemma stays the same in both cases.

   In the call-by-name situation we must take care, however, that in the translation $[\![\langle \mu\alpha.\langle v\{E_1\} \parallel e\{E_2\}\rangle \parallel E\{E_3\}\rangle]\!] = \langle \mu\alpha.\langle \textsc{subs}(v, \{E_1\}) \parallel \textsc{subs}(e, \{E_2\})\rangle \parallel \textsc{subs}(E, E_3)\rangle$, $\textsc{subs}(E, E_3)$ cannot become a $\tilde{\mu}$-abstraction. This is indeed the case because, for such a substitution to become a $\tilde{\mu}$-abstraction we must have that $E = \beta$ and $E_3(\beta) = \tilde{\mu}x.\langle v'\{E_4\} \parallel e'\{E_5\}\rangle$. This situation, however, cannot occur because that would mean that we encountered a state $\langle \mu\beta.\langle v_1\{E_v\} \parallel e_1\{E_e\}\rangle \parallel \tilde{\mu}x.\langle v'\{E_4\} \parallel e'\{E_5\}\rangle\rangle$ and reduced it to $\langle v_1\{\beta \mapsto \tilde{\mu}x.\langle v'\{E_4\} \parallel e'\{E_5\}\rangle, E_v\} \parallel e_1\{\beta \mapsto \tilde{\mu}x.\langle v'\{E_4\} \parallel e'\{E_5\}\rangle, E_e\}\rangle$, which we could not have done because reduction $(4)'$ disallows that move.

   Similarly, in the call-by-value situation, we must take care that $\textsc{subs}(V, E_1)$ in $[\![\langle V\{E_1\} \parallel \tilde{\mu}x.\langle v\{E_2\} \parallel e\{E_3\}\rangle\rangle]\!] = \langle \textsc{subs}(V, \{E_1\}) \parallel \tilde{\mu}x.\langle \textsc{subs}(v, \{E_2\}) \parallel \textsc{subs}(e, \{E_3\})\rangle\rangle$ does not become a $\mu$-abstraction. This is indeed the case because, for such a substitution to become a $\mu$-abstraction we must have that $V = y$ and $E_1(y) = \mu\alpha.\langle v'\{E_4\} \parallel e'\{E_5\}\rangle$. This situation cannot occur because that would mean that we encountered a state $\langle \mu\alpha.\langle v'\{E_4\} \parallel e'\{E_5\}\rangle \parallel \tilde{\mu}y.\langle v_1\{E_v\} \parallel e_1\{E_e\}\rangle\rangle$ and reduced it to $\langle v_1\{y \mapsto \mu\alpha.\langle v'\{E_4\} \parallel$

$e'\{E_5\}\rangle, E_v\} \parallel e_1\{y \mapsto \mu\alpha.\langle v'\{E_4\} \parallel e'\{E_5\}\rangle, E_e\}\rangle$, which we could not have done because reduction $(5)'$ disallows that move.

In the completeness lemma (Lemma 44) we must now use Lemma 38 to prove that the machine cannot run forever.

□

**Example 19.** Let us analyse how the term $t = \mu\alpha.\langle I^x \parallel \mu\beta.\langle I^y \parallel I^z \cdot \beta\rangle \cdot \alpha\rangle$ reduces in both the call-by-name and call-by-value machine. We begin with the call-by-name case, where:

$$\{E_1\} = \{\beta \mapsto \alpha\{x \mapsto (\mu\beta.\langle I^y\{\} \parallel (I^z \cdot \beta)\{\}\rangle)\}\}$$
$$\{E_2\} = \{y \mapsto I^z\{E_1\}, E_1\}$$

$$\begin{aligned}
&\mu\alpha.\langle I^x\{\} \parallel (\mu\beta.\langle I^y \parallel I^z \cdot \beta\rangle \cdot \alpha)\{\}\rangle \\
&\to \mu\alpha.\langle \mu\beta.\langle I^y\{\} \parallel (I^z \cdot \beta)\{\}\rangle \parallel \tilde{\mu}x.\langle x\{\} \parallel \alpha\{\}\rangle\rangle \\
&\to \mu\alpha.\langle x\{x \mapsto \mu\beta.\langle I^y\{\} \parallel (I^z \cdot \beta)\{\}\rangle\} \parallel \alpha\{x \mapsto \mu\beta.\langle I^y\{\} \parallel (I^z \cdot \beta)\{\}\rangle\}\rangle \\
&\to \mu\alpha.\langle \mu\beta.\langle I^y\{\} \parallel (I^z \cdot \beta)\{\}\rangle \parallel \alpha\{x \mapsto (\mu\beta.\langle I^y\{\} \parallel (I^z \cdot \beta)\{\}\rangle)\}\rangle \\
&\to \mu\alpha.\langle I^y\{E_1\} \parallel (I^z \cdot \beta)\{E_1\}\rangle \\
&\to \mu\alpha.\langle I^z\{E_1\} \parallel \tilde{\mu}y.\langle y\{E_1\} \parallel \beta\{E_1\}\rangle\rangle \\
&\to \mu\alpha.\langle y\{E_2\} \parallel \beta\{E_2\}\rangle \\
&\to \mu\alpha.\langle I^z\{E_1\} \parallel \beta\{E_2\}\rangle \\
&\to \mu\alpha.\langle I^z\{E_1\} \parallel \alpha\{x \mapsto (\mu\beta.\langle I^y\{\} \parallel (I^z \cdot \beta)\{\}\rangle)\}\rangle
\end{aligned}$$

In the call-by-value case we get the following reduction, where:

$$\{E_1\} = \{\beta \mapsto \tilde{\mu}x.\langle x\{\} \parallel \alpha\{\}\rangle\}$$
$$\{E_2\} = \{y \mapsto I^z\{E_1\}, E_1\}$$
$$\{E_3\} = \{x \mapsto I^z\{E_1\}\}$$

$$\begin{aligned}
&\mu\alpha.\langle I^x\{\} \parallel (\mu\beta.\langle I^y \parallel I^z \cdot \beta\rangle \cdot \alpha)\{\}\rangle \\
&\to \mu\alpha.\langle \mu\beta.\langle I^y\{\} \parallel (I^z \cdot \beta)\{\}\rangle \parallel \tilde{\mu}x.\langle x\{\} \parallel \alpha\{\}\rangle\rangle \\
&\to \mu\alpha.\langle I^y\{E_1\} \parallel (I^z \cdot \beta)\{E_1\}\rangle \\
&\to \mu\alpha.\langle I^z\{E_1\} \parallel \tilde{\mu}y.\langle y\{E_1\} \parallel \beta\{E_1\}\rangle\rangle \\
&\to \mu\alpha.\langle y\{E_2\} \parallel \beta\{E_2\}\rangle \\
&\to \mu\alpha.\langle I^z\{E_1\} \parallel \beta\{E_2\}\rangle \\
&\to \mu\alpha.\langle I^z\{E_1\} \parallel \tilde{\mu}x.\langle x\{\} \parallel \alpha\{\}\rangle\rangle \\
&\to \mu\alpha.\langle x\{E_3\} \parallel \alpha\{E_3\}\rangle \\
&\to \mu\alpha.\langle I^z\{E_1\} \parallel \alpha\{E_3\}\rangle
\end{aligned}$$

# Chapter 5

# Conclusion

In Chapter 3 we recalled Herbelin's $\bar{\lambda}\mu$-calculus [Her95]. We have defined the inverse of the map $\mathcal{N}$, which was defined in [CH00], and showed that both mappings preserve reduction and typing. This shows that $\bar{\lambda}\mu$ is, indeed, isomorphic to Parigot's $\lambda\mu$, as noted in [CH00]. As a consequence of this isomorphism, we were able to define an exception handling mechanism for $\bar{\lambda}\mu$ similar to the one defined in e.g. [Kre10] for $\lambda\mu$. Our main contribution to $\bar{\lambda}\mu$ is given in Section 2.2.2, where we showed that a restriction of the abstract machine defined in [CH00] is correct and complete for the weak head reduction strategy in $\bar{\lambda}\mu$.

In Chapter 4 we extended $\bar{\lambda}\mu$ with the $\tilde{\mu}$ operator and thereby obtained the $\bar{\lambda}\mu\tilde{\mu}$-calculus, which was first introduced by Herbelin and Curien in [CH00] (also see [Her05]). We showed, in Section 4.1, that $\bar{\lambda}\mu$'s exception handling mechanism can also be used for $\bar{\lambda}\mu\tilde{\mu}$. We noted that an extra reduction can be obtained for the mechanism, since we can perform call-by-value reduction steps in $\bar{\lambda}\mu\tilde{\mu}$ (which we cannot do in $\bar{\lambda}\mu$). We concluded the chapter by showing that Herbelin and Curien's (non-deterministic) machine [CH00] is correct and complete for the (non-deterministic) weak head reduction system in $\bar{\lambda}\mu\tilde{\mu}$. We also showed that the non-determinism in both machine and (weak head) reduction system can be solved, by which we obtain, correct and complete, call-by-name and call-by-value specific versions of the machine and strategy.

## 5.1 Future works

The machines presented in this thesis use quite big reduction steps - compared to e.g. a Krivine machine - in that arguments are not pushed to and popped from a stack. While the syntax of $\bar{\lambda}\mu$ and $\bar{\lambda}\mu\tilde{\mu}$ allow for these big steps, it would be interesting to develop machines that do push and pop arguments from a stack, as this is closer to how "real" machines work.

In Section 7 of [CH00], Herbelin and Curien further extend the $\bar{\lambda}\mu\tilde{\mu}$-calculus with the term-construct $e \cdot v$ (which is the dual of the context-construct $v \cdot e$) and the context-construct $\beta\lambda.e$ (which is the dual of the term-construct $\lambda x.v$). On the logical side these extensions correspond the the right- and left introduction of "-", the dual of $\rightarrow$. We have the following typing rules for $-$:

$$\frac{\Gamma \mid e : A \vdash \beta : B, \Delta}{\Gamma \mid \beta\lambda.e : A - B \vdash \Delta} -_l \quad \frac{\Gamma \mid e : B \vdash \Delta \quad \Gamma \vdash v : A \mid \Delta}{\Gamma \vdash (e \cdot v) : A - B \mid \Delta} -_r$$

On the computational side we obtain the following reduction:

$$\langle e_2 \cdot v \parallel \beta\lambda.e_1 \rangle \to_- \langle \mu\beta.\langle v \parallel e_1 \rangle \parallel e_2 \rangle$$

While Herbelin and Curien treat the $-$ connective in a purely formal way, Crolard [Cro04] has initiated research into the computational meaning of $-$: it provides us with a way to express coroutines. It would be interesting to develop a machine capable of evaluating these new language constructs and thereby obtain an operational semantics for the $-$ connective.

# Bibliography


[BDS12] Henk Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Perspectives in Mathematical Logic. Cambridge University Press, 2012.

[CH00] Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000*, SIGPLAN Notices 35(9), pages 233–243. ACM, 2000.

[Chu40] A. Church. A formulation of the simple theory of types. *The journal of symbolic logic*, 5(2):56–68, 1940.

[Cro99] T. Crolard. A confluent $\lambda$-calculus with a catch/throw mechanism. *Journal of Functional Programming*, 9(6):625–647, 1999.

[Cro04] T. Crolard. A formulae-as-types interpretation of subtractive logic. *Journal of Logic and Computation*, 14(4):529–570, 2004.

[DF07] R. Douence and P. Fradet. The next 700 krivine machines. *Higher-Order and Symbolic Computation*, 20(3):237–255, 2007.

[DG98] P. De Groote. An environment machine for the $\lambda\mu$-calculus. *Mathematical Structures in Computer Science*, 8(6):637–669, 1998.

[FH92] M. Felleisen and R. Hieb. The revised report on the syntactic theories of sequential control and state. *Theoretical computer science*, 103(2):235–271, 1992.

[Fil89] A. Filinski. Declarative continuations: An investigation of duality in programming language semantics. In *Category Theory and Computer Science*, pages 224–249. Springer, 1989.

[Gri89] T.G. Griffin. A formulae-as-type notion of control. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 47–58. ACM, 1989.

[GTL89] J.Y. Girard, P. Taylor, and Y. Lafont. *Proofs and types*, volume 7. Cambridge University Press Cambridge, 1989.

[Her95] Hugo Herbelin. *Séquents qu'on calcule: de l'interprétation du calcul des séquents comme calcul de $\lambda$-termes et comme calcul de stratégies gagnantes*. Ph.D. thesis, University Paris 7, January 1995.

[Her05] H. Herbelin. C'est maintenant qu'on calcule: au cœur de la dualité. *Mémoire d'habilitation, available from http://pauillac.inria.fr/∼herbelin/habilitation/*, 2005.

[Her10] H. Herbelin. An intuitionistic logic that proves markov's principle. In *Logic in Computer Science (LICS), 2010 25th Annual IEEE Symposium on*, pages 50–56. IEEE, 2010.

[HZ09] H. Herbelin and S. Zimmermann. An operational account of call-by-value minimal and classical $\lambda$-calculus in "natural deduction" form. *Typed Lambda Calculi and Applications*, pages 142–156, 2009.

[Kre10] Robbert Krebbers. Classical logic, control calculi and data types. Master's thesis, Radboud University Nijmegen, 2010.

[Kri07] J.L. Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, 2007.

[Lan64] P.J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, 1964.

[MOW98] J. Maraist, M. Odersky, and P. Wadler. The call-by-need lambda calculus. *Journal of functional programming*, 8(03):275–317, 1998.

[Nak92] H. Nakano. A constructive formalization of the catch and throw mechanism. In *Logic in Computer Science, 1992. LICS'92., Proceedings of the Seventh Annual IEEE Symposium on*, pages 82–89. IEEE, 1992.

[Nak03] K. Nakazawa. Confluency and strong normalizability of call-by-value $\lambda\mu$-calculus. *Theoretical Computer Science*, 290(1):429–463, 2003.

[Par92] M. Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In *Logic programming and automated reasoning*, pages 190–201. Springer, 1992.

[Par97] M. Parigot. Proofs of strong normalisation for second order classical natural deduction. *Journal of Symbolic Logic*, pages 1461–1479, 1997.

[Plo75] G.D. Plotkin. Call-by-name, call-by-value and the $\lambda$-calculus. *Theoretical computer science*, 1(2):125–159, 1975.

[RS94] Jakob Rehof and Morten Heine Sørensen. The lambdadelta-calculus. In *Proceedings of the International Conference on Theoretical Aspects of Computer Software*, TACS '94, pages 516–542, London, UK, UK, 1994. Springer-Verlag.

[Sel01] 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.

[Tak89] Masako Takahashi. Parallel reductions in $\lambda$-calculus. *Journal of Symbolic Computation*, 7(2):113 – 123, 1989.