Functions as processes: termination and the $\bar{\lambda}\mu\tilde{\mu}$-calculus[*]

Matteo Cimini[1], Claudio Sacerdoti Coen[2], and Davide Sangiorgi[2,3]

[1] School of Computer Science, Reykjavik University, Iceland
[2] Department of Computer Science, University of Bologna, Italy
[3] INRIA, France

Abstract. The $\bar{\lambda}\mu\tilde{\mu}$-calculus is a variant of the λ-calculus with significant differences, including non-confluence and a Curry-Howard isomorphism with the classical sequent calculus.

We present an encoding of the $\bar{\lambda}\mu\tilde{\mu}$-calculus into the π-calculus. We establish the operational correctness of the encoding, and then we extract from it an abstract machine for the $\bar{\lambda}\mu\tilde{\mu}$-calculus. We prove that there is a tight relationship between such a machine and Curien and Herbelin's abstract machine for the $\bar{\lambda}\mu\tilde{\mu}$-calculus. The π-calculus image of the (typed) $\bar{\lambda}\mu\tilde{\mu}$-calculus is a nontrivial set of terminating processes.

1 Introduction

In his seminal paper [10], Milner gave accurate compilations from the call-by-value and call-by-name λ-calculi into the π-calculus. The study of embeddings of λ-calculi into a process calculus has then been continued, by a number of researchers (see [14] for a tutorial), and is interesting for several reasons. From the process calculus point of view, it is a significant test of expressiveness, and helps in getting deeper insight into its theory. From the λ-calculus point of view, it provides the means to study λ-terms in contexts other than purely sequential ones, and with the instruments available in the process calculus. For example, an important behavioural equivalence upon process terms gives rise to an interesting equivalence upon λ-terms. Moreover, the relevance of those λ-calculus evaluation strategies which can be efficiently encoded is strengthened. The study can also be useful to provide a semantic foundation for languages which combine concurrent and functional programming and to develop parallel implementations of functional languages.

Last but not least, the study can give insights into the transfer of results or techniques from the λ-calculus into the process calculus. An example are results about termination of programs. Termination has been studied extensively in the λ-calculus, where it is often called *strong normalisation*. Termination is also important in concurrency. For instance, if we interrogate a server, we may want to know that the interrogation does not cause an infinite computation in the server. Compared to the λ-calculus, results about

---

termination in the π-calculus are fairly rare [20, 15, 6, 5]. Other interesting results in the λ-calculi concern the Curry-Howard isomorphism, in the sense of [7, 17], between the formulae of a logic and the the types of a calculus. An example of a technique widely used in the λ-calculus and that would be interesting to transport onto the π-calculus are logical relations. The technique is used in the λ-calculus to establish various properties including termination, parametricity and representation independence. It remains rather unclear how to transport the technique onto a concurrent language so to capture also processes that are 'non-functional' and non-confluent.

In the present work, we explore the embedding into π-calculus of a variant of the λ-calculus, namely the $\bar{\lambda}\mu\tilde{\mu}$-calculus [4]. The $\bar{\lambda}\mu\tilde{\mu}$-calculus presents some striking differences with respect to the ordinary λ-calculus. A major interest of the $\bar{\lambda}\mu\tilde{\mu}$-calculus is that its typed version is Curry-Howard isomorphic to classical sequent calculus. While in the present work we focus on the untyped version, the Curry-Howard correspondence is still reflected in the untyped calculus in several ways. We discuss all these aspects below.

First of all, the $\bar{\lambda}\mu\tilde{\mu}$-calculus, being Curry-Howard isomorphic to a sequent calculus (as the simpler $\bar{\lambda}$-calculus of Herbelin [8]), can be seen as a calculus whose terms are reduction machine states. In particular, and contrary to the λ-calculus, all head reductions only involve the outermost parts of the term. In the λ-calculus, even in the case of head reduction, the parts of the λ-term that interact need not be at the outermost level, as shown in the term $((\lambda x.\, M)N)P$. Encodings of the λ-calculus into π-calculus have to bring the interacting terms to a topmost position to allow interaction between them. This forces, for instance, the encoding of λ-terms to be parametric on the channels used to interact, which have to be provided dynamically. In the $\bar{\lambda}\mu\tilde{\mu}$-calculus, in contrast, a redex is always at top level. For instance, the previous example becomes $\langle \lambda x.\, M \mid N \cdot (P \cdot \alpha) \rangle$: the λ-abstraction and its argument N are in the outermost parts of the term. The topmost property allows us to avoid the channel parametrization in the encoding into π-calculus: only three channels are needed for reduction, each one corresponding to a different kind of redex.

Secondly, the $\bar{\lambda}\mu\tilde{\mu}$-calculus is strongly normalizing, but non-confluent, as required by Girard's example of non-confluence for cut elimination in classical logic. Thus, one may hope that the subset of the π-calculus obtained as the image of the encoding, restricted to well-typed terms, is a nontrivial example of a non-confluent but strongly normalizing set of processes. In order to achieve non-confluence, the $\bar{\lambda}\mu\tilde{\mu}$-calculus includes the μ control operator that behaves like the μ control operator of Parigot's $\lambda\mu$-calculus [12], introducing a critical pair in the reduction system. The critical pair, however, is only a symptom of a more interesting phenomenon: each topmost subterm (that becomes a process in π-calculus) can always express two different behaviours: either it can be bound and delayed, acting in a passive way, or it can continue its normal behaviour, for instance by capturing another topmost subterm. Moreover, the subterm is actually captured only if it interacts with a capturing term. Since each one is a context for the other and determines the other subterm behaviour, the subterms involved in a reduction interact in an essentially synchronous way. We can see indeed the (encoding of the) $\bar{\lambda}\mu\tilde{\mu}$-calculus as a calculus of peers where each peer can initiate the communication, whereas the λ-calculus (with a fixed strategy) follows the client-server model.

These synchrony aspects explain why our encoding of the $\bar{\lambda}\mu\tilde{\mu}$-calculus exploits the synchronous features of the π-calculus, notably mixed choice, i.e. a choice between an input and an output action.

A finally reason of interest for the $\bar{\lambda}\mu\tilde{\mu}$-calculus is that the application of the logical relation technique to it is technically quite different from that for the ordinary λ-calculus. This is partly due to the non-confluence properties of the $\bar{\lambda}\mu\tilde{\mu}$-calculus and partly, but more significantly, to its control operators. These aspects make us believe that the logical relations in the $\bar{\lambda}\mu\tilde{\mu}$-calculus can offer insights for the transport of logical relations onto a concurrent calculus such as the π-calculus, though we do not pursue this line of work in the present paper.

Figure 1 is a summary of the main results. Following the work of Vasconcelos in [19], we identify the reduction machine $M1$ implemented by the encoding and we put it in relation with a previously known reduction machine for the $\bar{\lambda}\mu\tilde{\mu}$-calculus given by Curien and Herbelin in [4], here called $M2$. The operational behaviour of the encoding π-terms is precisely captured by the reduction machine $M1$ (derived in a straightforward way from the $\bar{\lambda}\mu\tilde{\mu}$-to-$\pi$ encoding). This machine $M1$ is essentially equivalent to $M2$. Hence the diagram in the figure commutes and all encodings are correct and complete. Moreover, we also show that some encodings (those marked '1-1,1-1') are *strong operational correspondences*, i.e. the encoding is bijective and every reduction step of the encoded term corresponds exactly to one step of the encoding term and vice versa.

Besides the synchronous encoding, we also present an encoding into the asynchronous π-calculus. As expected, we need to resolve the mixed choice by introducing an arbiter, which yields an heavier encoding. The arbiter is responsible for breaking the critical pair, and it can be biased in order to always give precedence to one side of the pair. Being able to consistently resolve the critical pairs allows to exploit Curien and Herbelin's two encodings of the λ-calculus into the $\bar{\lambda}\mu\tilde{\mu}$-calculus that yields respectively a call-by-value and a call-by-name reduction.
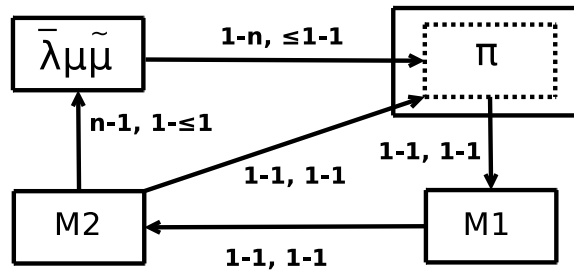
Proofs and examples are here omitted and can be found in [3].

Structure of the paper In Section 2 we recall the definitions of the $\bar{\lambda}\mu\tilde{\mu}$-calculus and of the Curien and Herbelin's abstract machine. In Section 3 we provide the encoding to π-calculus. Section 4 is devoted to presenting the machine $M1$, whose relationship to the encoding is then addressed in Section 5. In Section 6 we finally prove the correspondence between $M1$ and $M2$, making the diagram of Figure 1 commute. In Section 7 we provide an asynchronous version of the encoding. Conclusions and future work are discussed in Section 8.

2 Preliminaries

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

Assuming familiarity with the λ-calculus [1], in this section we recall the definitions of the $\bar{\lambda}\mu\tilde{\mu}$-calculus. A precise description of the $\bar{\lambda}\mu\tilde{\mu}$-calculus is out of the scope of this paper and can be found in [4]. The calculus is characterized by three syntactic categories: terms, contexts (dual to terms) and commands. Syntax and the operational semantics are defined as follows.

1-n, ≤1-1	Correctness: - if $M \to N$ then $[\![M]\!] \to N'$ where $(N, N') \in \mathcal{R}$ - if $[\![M]\!] \to N$ then $\exists N', M \to^{\leq 1} N' \wedge (N', N) \in \mathcal{R}$ where $\mathcal{R}$ is $1 - n$ and modelled after Milner [10]
n-1, 1-≤ 1	Correctness: - if $M \to N$ then $[\![M]\!] \to^{\leq 1} [\![N]\!]$ - if $[\![M]\!] \to N$ then $\exists N', M \to N' \wedge [\![N']\!] = N$ where $[\![\cdot]\!] = \cdot$ is $n - 1$ and equal to $\mathcal{R}^{-1}$
1-1, 1-1	Strong Operational Correspondence: - $M \to N$ iff $[\![M]\!] \to [\![N]\!]$

Fig. 1. The main results on the embedding of $\bar{\lambda}\mu\tilde{\mu}$-terms into π-terms

Syntax:		Reduction rules:
$Command\ c = \langle v \mid e \rangle$	$\lambda - reduction:$	$\langle \lambda x.\, v_1 \mid v_2 \cdot e \rangle \rightarrow \langle v_2 \mid \tilde{\mu}x.\, \langle v_1 \mid e \rangle \rangle$
$Term\ v = x \mid \lambda x.\, v \mid \mu\beta.\, c$	$\mu - reduction:$	$\langle \mu\beta.\, c \mid e \rangle \rightarrow c[e/\beta]$
$Context\ e = \alpha \mid v \cdot e \mid \tilde{\mu}x.\, c$	$\tilde{\mu} - reduction:$	$\langle v \mid \tilde{\mu}x.\, c \rangle \rightarrow c[v/x]$

A command is a closed system containing both the program and the context where it is evaluated. Besides the usual λ-abstraction the calculus has few constructs which deserve some words:

- $\tilde{\mu}x.\, c$: The (context or continuation) variable β is bound in c. Semantically, the $\tilde{\mu}$ operator is the standard local definition operator of ML: $\langle \cdot \mid \tilde{\mu}x.\, c \rangle$ is equivalent to `let` $x := \cdot$ `in` c; operationally, it can capture the term in dot position and substitute it in place of x in c, actually delaying its evaluation in a call-by-name fashion.
- $\mu\beta.\, c$: The (term) variable x is bound in c. Dually to $\tilde{\mu}$, the μ operator binds its evaluation context (like Parigot's μ operator [12] and similarly to the call/cc operator of Scheme) and, operationally, it can capture the context substituting it in place of β in c, actually delaying its evaluation and bringing c in topmost position, in a call-by-value fashion.
- $v \cdot e$: Called the *cons operator*, it is a context for a λ-abstraction that feeds the input v to the λ-abstraction and collects the result of the evaluation by matching it against the context e; operationally, a β-redex is not reduced by performing an immediate substitution (that would correspond to call-by-name): the argument is put in head position and matched by a $\tilde{\mu}$-context that can be fired (like in call-by-name) or that can be delayed (like in call-by-value, if the argument becomes a μ-term).

The reduction takes place at the topmost position, performing the so called *topmost reduction*. This is consistent with the game-theoretic view of the calculus where terms and contexts are players interacting each other in order to perform a computational step. The two players are at the top of the abstract syntax tree and a reduction step is responsible to make them vanish and bring (possibly from the lower levels) the new term and the new context ready to interact again.

As mentioned in the introduction, the calculus is non-confluent. Commands of the form $\langle \mu\beta.\, c_1 \mid \tilde{\mu}x.\, c_2 \rangle$ allow indeed for both μ-reduction and $\tilde{\mu}$-reduction to take place and may result into two different reducts:

$$\langle \mu\beta.\, c_1 \mid \tilde{\mu}x.\, c_2 \rangle \rightarrow c_1[\tilde{\mu}x.\, c_2/\beta]$$

$$\langle \mu\beta.\, c_1 \mid \tilde{\mu}x.\, c_2 \rangle \rightarrow c_2[\mu\beta.\, c_1/x]$$

with $c_1[\tilde{\mu}x.\, c_2/\beta]$ and $c_2[\mu\beta.\, c_1/x]$ possibly distinct normal forms.

Finally, it is worth noting the dualities between $\mu/\tilde{\mu}$ and term/context variables. In [4] particular attention is devoted to such dualities. Discussing them in detail is out of the scope of this paper and the interested reader is invited to refer to [4].

2.2 The reduction machine $M2$

In [4], Curien and Herbelin also provide a reduction machine for the $\bar{\lambda}\mu\tilde{\mu}$-calculus. Before embarking in the definition of such a machine, the formal notion of reduction machine from [19] is repeated here.

Definition 1 (Reduction Machine). *A reduction machine is a triple $\langle S, \to^s, \approx^s \rangle$, where S is a set (the set of states), $\to^s \subset S \times S$ (the reduction relation), and $\approx^s$ is an equivalence relation in $S \times S$, such that $\approx^s \to^s \ \subseteq \ \to^{*s} \approx^s$.*

Definition 1 slightly differs from [19] for the author treats exclusively deterministic machines. We have just dropped the deterministic clause, keeping the name.

The mentioned Curien and Herbelin's machine, here called $M2$ for simplicity, is presented below.

Definition 2 (States and Environments of the machine $M2$).

$$
\begin{aligned}
State \quad s &= \langle vcl \mid ecl \rangle \\
Term\ Closure \quad vcl &= v\ in\ \rho \\
Context\ Closure \quad ecl &= e\ in\ \rho \\
Environment \quad \rho &= \emptyset \mid [var_v = vcl] + \rho \mid [var_e = ecl] + \rho
\end{aligned}
$$

with var_v and var_e term and context variables, respectively. We write S_{M2} to refer to the set of the states of the machine $M2$.

The operational behaviour of the machine $M2$ is described by means of the reduction relation $\longrightarrow^{M2} \subset S_{M2} \times S_{M2}$.

Definition 3 (The reduction relation $\longrightarrow^{M2}$).

$$
\begin{aligned}
\langle \mu\beta.\, c\ in\ \rho_1 \mid e\ in\ \rho_2 \rangle &\longrightarrow^{M2} c\ in\ \rho_1 + [\beta = e\ in\ \rho_2] \\
\langle v\ in\ \rho_1 \mid \tilde{\mu}x.\, c\ in\ \rho_2 \rangle &\longrightarrow^{M2} c\ in\ \rho_2 + [x = v\ in\ \rho_1] \\
\langle x\ in\ \rho_1 + [x = v\ in\ \rho_2] \mid e\ in\ \rho_3 \rangle &\longrightarrow^{M2} \langle v\ in\ \rho_2 \mid e\ in\ \rho_3 \rangle \\
\langle v\ in\ \rho_1 \mid \beta\ in\ \rho_2 + [\beta = e\ in\ \rho_3] \rangle &\longrightarrow^{M2} \langle v\ in\ \rho_1 \mid e\ in\ \rho_3 \rangle \\
\langle \lambda x.\, v_1\ in\ \rho_1 \mid v_2 \cdot e\ in\ \rho_2 \rangle &\longrightarrow^{M2} \langle v_1\ in\ \rho_1 + [x = v_2\ in\ \rho_2] \mid e\ in\ \rho_2 \rangle
\end{aligned}
$$

The first four rules are simply obtained by replacing immediate substitution with delayed substitution, implemented by means of local explicit substitutions for the topmost term and context. The last rule is more involved: in order to maintain the invariant that the local substitution is applied only to topmost terms and contexts, the λ-reduction rule is twisted a bit. In the rest of the paper we consider a variant of the reduction machine obtained by dropping the twisted rule and by replacing it with the following one, that captures λ-reduction more closely at the price of allowing explicit substitutions on a subterm:

$$
\langle \lambda x.\, v_1\ in\ \rho_1 \mid v_2 \cdot e\ in\ \rho_2 \rangle \longrightarrow^{M2} \langle v_2\ in\ \rho_2 \mid \tilde{\mu}x.\, \langle v_1\ in\ \rho_1 \mid \delta\ in\ [\delta = e\ in\ \rho_2] \rangle \rangle
$$

States in S_{M2} are considered up-to the following equivalence relation $\equiv^{M2}$.

Definition 4 (The equivalence relation $\equiv^{M2}$).

$$t \text{ in } \rho_1 + [var_1 = cl_1] + [var_2 = cl_2] + \rho_2 \equiv^{M2} t \text{ in } \rho_1 + [var_2 = cl_2] + [var_1 = cl_1] + \rho_2$$

$$t \text{ in } \rho_1 + [var = cl] + \rho_2 \equiv^{M2} t[var'/var] \text{ in } \rho_1 + [var' = cl] + \rho_2$$
$$\text{if } var' \notin FV(t)$$

$$t \text{ in } \rho_1 + [var = cl] + \rho_2 \equiv^{M2} t \text{ in } + \rho_1 + \rho_2 \quad \text{if } var \notin FV(t)$$

Intuitively, the equivalence relation $\equiv^{M2}$ identifies explicit substitutions up to commutativity, α-equivalence of environment entry names and garbage collection of unused substitutions.

The reduction machine $M2$ is the triple $\langle S_{M2}, \rightarrow^{M2}, \equiv^{M2} \rangle$.

3 From $\bar{\lambda}\mu\tilde{\mu}$ to π

We assuming the reader familiar with the π-calculus [11, 16]. We encode the $\bar{\lambda}\mu\tilde{\mu}$-commands into the dialect of the π-calculus with internal mobility, namely the πI-calculus [13], where only the output of fresh new channels is allowed. We employ such a calculus admitting both replication $!P$ and recursion $Rec\ X.\ P$. The encoding is the following one, the reader should bear in mind that, as usual in the πI-calculus, we write $\bar{x}(y).\ P$ for $\nu y\,(\bar{x}\langle y\rangle.\ P)$.

$$[\![\langle v \mid e\rangle]\!] = [\![v]\!] \mid [\![e]\!]$$
$$[\![\mu\beta.\ c]\!] = Rec\ Y.\ (\overline{\mu}(\beta).\ [\![c]\!] + \tilde{\mu}(x).\ !x.\ Y)$$
$$[\![\tilde{\mu}x.\ c]\!] = Rec\ Y.\ (\overline{\tilde{\mu}}(x).\ [\![c]\!] + \mu(\beta).\ !\beta.\ Y)$$
$$[\![x]\!] = Rec\ Y.\ (\overline{x} + \tilde{\mu}(z).\ !z.\ Y)$$
$$[\![\beta]\!] = Rec\ Y.\ (\overline{\beta} + \mu(\beta).\ !\beta.\ Y)$$
$$[\![\lambda x.\ v_1]\!] = Rec\ Y.\ (\lambda(\delta).\ [\![\tilde{\mu}x.\ \langle v_1 \mid \delta\rangle]\!] + \tilde{\mu}(z).\ !z.\ Y)$$
$$[\![v_2 \cdot e]\!] = Rec\ Y.\ (\overline{\lambda}(\delta).\ ([\![v_2]\!] \mid !\delta.\ [\![e]\!]) + \mu(\beta).\ !\beta.\ Y)$$

The encoding of a command is simply the parallel composition between the encoding of the topmost term and its context, according to the game-theoretic reading of the calculus where every player becomes a process. This property is disrupted in the asynchronous encoding we present in Section 7, which is based on an arbiter.

As for the λ-calculus, substitution is immediate in the $\bar{\lambda}\mu\tilde{\mu}$-calculus, but needs to be delayed both in reduction machines (to achieve efficiency) and in encodings into first order process calculi. We achieve this using the standard technique, already used by Milner, of binding terms by creating fresh channel names and activating them lazily by interacting on the channel. Differently from Milner, however, we let the binding term generate the fresh channel, reusing the variable name coming from the λ-term and thus avoiding to parameterize the encoding or to use syntactical substitution operators.

The encoding uses three selected channels, named μ, $\tilde{\mu}$ and λ to mimic the corresponding redexes. The μ-reduction is implemented by letting every context encoding be

of the form *Rec Y*. $(\cdot + \mu(\beta). \,!\beta. \,Y)$ and letting the encoding of $\mu\beta. \,c$ fire a fresh channel β over μ to its context, activating the right part of the mixed choice and delaying the execution of Y that waits over β for (multiple) activation. A perfectly dual solution is used to mimic $\tilde{\mu}$-reduction, leading to the critical pair obtained by the translation of $[\![\langle \mu\beta. \,c \mid \tilde{\mu}x. \,c'\rangle]\!]$ where two mixed choices can interact both on the μ and the $\tilde{\mu}$ channels to perform either μ-reduction or $\tilde{\mu}$-reduction. In a precise sense, we can see the (encoding of the) $\bar{\lambda}\mu\tilde{\mu}$-calculus as a calculus of peers where each peer can initiate the communication, whereas the λ-calculus (with a fixed strategy) follows the client-server model. It is thus not surprising that the encoding requires a synchronous calculus.

The λ-reduction is more involved since $[\![v_1]\!]$ and $[\![e]\!]$, that belong to different processes, must behave as $[\![\langle v_1 \mid e\rangle]\!]$ where $\langle v_1 \mid e\rangle$ is not a sub-term of the original process, and neither e is accessible in the encoding of $\lambda x. \,v_1$ nor the other way around. We solve the problem by handling $[\![e]\!]$ as an argument, guarding its execution by δ and sending δ to the encoding of $\lambda x. \,v_1$ by means of the channel λ.

The reader may have noticed that having both replication and recursion, albeit unusual and redundant, captures in a natural manner two different behaviours of the processes: bound processes waiting to be activated will be replicated; processes in mixed choice that need to receive a channel and replicate their behaviour on that channel need to be implemented using recursion.

In [3] we provide an example of encoding and we show its reduction steps. The reader can find in [3] also the proof of the correctness of the encoding (the right-headed arrow in Figure 1). In order to establish this result we employ the same technique already used by Milner in [10], that consists in relating (via a relation $\mathcal{R}$) each $\bar{\lambda}\mu\tilde{\mu}$-term M not only to $[\![M]\!]$, but to a larger family of π-terms that are all representations of M up to "delayed substitutions". Indeed, since the π-calculus is first order, the encoding cannot capture precisely immediate substitution, and thus it captures an explicit form of delayed substitution. Thus, during reduction, the same $\bar{\lambda}\mu\tilde{\mu}$-term M receives multiple representations that are all π-processes that are not observationally equivalent, but that can still be "identified" by relating them via $\mathcal{R}$ to M. Formally, $\mathcal{R}$ is the minimal one-to-many relation such that $(M, P) \in \mathcal{R}$ for all M, P, M', N_i, x_i such that $M = M'[N_1/x_1; \ldots; N_n/x_n]$ and $P = \nu x_1, x_2, \ldots, x_n \,([\![M']\!] \mid \,!x_1. \,[\![N_1]\!] \mid \,\ldots \,\mid \,!x_n. \,[\![N_n]\!])$.

A few points are worth a mention:

– The dualities mentioned in Section 2.1 are preserved. The reader may notice that the encodings concerning μ- and $\tilde{\mu}$-abstractions, as well as the ones concerning term and context variables, are the same modulo the exchange of the channels μ and $\tilde{\mu}$.
– Contrary to the λ-calculus, the encoding is not parametrized.
– The synchronous nature of the $\bar{\lambda}\mu\tilde{\mu}$-calculus discussed in Section 2.1 is reflected by the employment of a synchronous calculus, which uses mixed choice.

4 The reduction machine induced from $[\![\cdot]\!]$

Following the work of Vasconcelos [19] we now question ourself about what reduction machine we are implicitly mimicking through the encoding $[\![\cdot]\!]$. In this section we present the machine $M1$ constructed by inspecting the way the compilation $[\![\cdot]\!]$ simulates the $\bar{\lambda}\mu\tilde{\mu}$-calculus.

By the considerations made in the introduction and at the end of Section 3, it is not surprising that $M1$ turns out to be an environment-based machine. The reader may consider the process

$$P = (\nu x, \beta)(\llbracket \langle x \mid \beta \rangle \rrbracket \mid {!}x.\llbracket v \rrbracket \mid {!}\beta.\llbracket e \rrbracket).$$

P represents the encoding of the command $\langle x \mid \beta \rangle$ when executed in an environment where x is bound to v and β is bound to e. We can think of P as $\llbracket \langle x \mid \beta \rangle \ in \ [x = v, \ \beta = e] \rrbracket$.

The states of the machine $M1$ are thus pairs consisting of a command and a global environment that maintains the bindings for its free variables. The states of the machine $M1$ are defined as follows.

Definition 5 (States and Environments of the machine $M1$).

$$State \ \ s \ = c \ in \ \rho$$
$$Environment \ \ \rho \ = \emptyset \mid [var_v = v] + \rho \mid [var_e = e] + \rho$$

with var_v and var_e term and context variables, respectively. We write S_{M1} to refer to the set of the states of the machine $M1$.

The operational behaviour of the machine $M1$ is described by means of the reduction relation $\longrightarrow^{M1} \subset S_{M1} \times S_{M1}$.

Definition 6 (The reduction relation $\longrightarrow^{M1}$).

$$\langle \mu\beta.c \mid e \rangle \ in \ \rho \longrightarrow^{M1} c[\beta'/\beta] \ in \ \rho \ + \ [\beta' = e]$$
$$\langle v \mid \tilde{\mu}x.c \rangle \ in \ \rho \longrightarrow^{M1} c[x'/x] \ in \ \rho \ + \ [x' = v]$$
$$\langle x \mid e \rangle \ in \ \rho \ + \ [x = v] \longrightarrow^{M1} \langle v \mid e \rangle \ in \ \rho \ + \ [x = v]$$
$$\langle v \mid \alpha \rangle \ in \ \rho \ + \ [\alpha = e] \longrightarrow^{M1} \langle v \mid e \rangle \ in \ \rho \ + \ [\alpha = e]$$
$$\langle \lambda x.v_1 \mid (v_2 \cdot e) \rangle \ in \ \rho \longrightarrow^{M1} \langle v_2 \mid \tilde{\mu}x.\langle v_1 \mid \delta \rangle \rangle \ in \ \rho \ + \ [\delta = e]$$
$$with \ \beta', \ x' \ and \ \delta \ fresh$$

To understand why a variable change is performed in the first two rules, the reader may consider the way in which the encoding simulates the μ-reductions. In the process $\llbracket \langle \mu\beta.c \mid e \rangle \rrbracket$, the encoding of the μ-abstraction sends a fresh new channel to the encoding of e, which we model by means of an α-conversion to a fresh variable.

States in S_{M1} are considered up-to the following equivalence relation $\equiv^{M1}$.

Definition 7 (The equivalence relation $\equiv^{M1}$).

$$c \ in \ \rho_1 + [var_1 = t_1] + [var_2 = t_2] + \rho_2 \equiv^{M1} c \ in \ \rho_1 + [var_2 = t_2] + [var_1 = t_1] + \rho_2$$
$$c \ in \ \rho_1 + [var = t] + \rho_2 \equiv^{M1} c[var'/var] \ in \ \rho_1 + [var' = t] + \rho_2$$
$$if \ var' \notin FV(c)$$
$$c \ in \ \rho_1 + [var = t] + \rho_2 \equiv^{M1} c \ in \ + \ \rho_1 + \rho_2 \quad if \ var \notin FV(c)$$

Intuitively, the equivalence relation $\equiv^{M1}$ identifies the states of the machine $M1$ up to commutativity of environment entries, α-equivalence of environment entry names and garbage collection of unused entries.

The reduction machine $M1$ is the triple $\langle S_{M1}, \rightarrow^{M1}, \equiv^{M1} \rangle$.

5 The relationship between encoded terms and $M1$

In the previous section we have extracted the reduction machine $M1$ from the encoding $[\![\cdot]\!]$ following simple intuitions concerning the way it operates. In this section we analyse the relationship between encoded π-calculus terms and terms of the machine $M1$, deferring to Section 6 the question whether $M1$ is a correct abstract machine for the $\bar{\lambda}\mu\tilde{\mu}$-calculus.

Intuitively, an operational correspondence holds whenever a reduction step in the source machine is mimicked by a sequence of steps in the target machine. In our setting we are able to establish a stronger correspondence between the execution of terms in the machine $M1$ and the behaviour of their encoding; we are able in fact to exhibit a 1-1 correspondence between the two.

We set up the notion of *strong operational correspondence*, summarized in Figure 1 and defined here more accurately.

Definition 8 (Strong Operational Correspondence). *Given the two reduction machines $S = \langle St_S, \rightarrow^s, \approx^s \rangle$ and $R = \langle St_R, \rightarrow^r, \approx^r \rangle$, a mapping $(\!|\cdot|\!) : St_S \rightarrow St_R$ is a strong operational correspondence between S and R whenever for all s and s' in St_S*

$$s \rightarrow^s s' \iff (\!|s|\!) \rightarrow^r (\!|s'|\!)$$

where states from S and R are considered equal up-to relations $\approx^s$ and $\approx^r$, respectively.

Proving a mapping $(\!|\cdot|\!)$ to be a strong operational correspondence is strongly demanding since R is required to step-by-step simulates S through the mapping $(\!|\cdot|\!)$.

In order to prove the strong operational correspondence between $M1$ states and π-terms, we need to make explicit the bijective mapping between $M1$ states and images in the π-calculus. The encoding $[\![\cdot]\!]$ is thus completed as follows:

$$[\![c \text{ in } \rho]\!] = \nu var_1, var_2, \ldots, var_n \, ([\![c]\!] \mid [\![\rho]\!])$$
$$\text{with } \rho = [var_1 = t_1, var_1 = t_2 \ldots var_n = t_n]$$
$$[\![\emptyset]\!] = \emptyset$$
$$[\![[var = t] + \rho]\!] = !var.[\![t]\!] \mid [\![\rho]\!]$$

As in the work of Vasconcelos [19], we need to set the π-fragment of Figure 1 as a reduction machine. We denote such a reduction machine by π, which consists of the triple $\langle \mathcal{P}, \rightarrow^\pi, \equiv^\pi \rangle$, where $\mathcal{P}$ is the codomain of $[\![\cdot]\!]$, $\rightarrow^\pi$ is the reduction relation of the π-calculus and $\equiv^\pi$ is the structural congruence, see [11, 16]. The following theorem ensures that encoded terms and states of the machine $M1$ are (operationally) intimately related to each other.

Theorem 1 (Strong Operational Correspondence between $M1$ and π). $[\![\cdot]\!]$ *is a strong operational correspondence between $M1$ and π, i.e for all s and s'*

$$s \;\to^{M1}\; s' \;\Leftrightarrow\; [\![s]\!] \;\to^{\pi}\; [\![s']\!]$$

where states from $M1$ and π are considered equal up-to relations $\equiv^{M1}$ and $\equiv^{\pi}$, respectively.

6 The relationship between $M1$ and $M2$

Clearly, the machine $M1$ and Theorem 1 are useful only when we prove that the machine $M1$ is correct w.r.t. the operational semantics of the $\bar{\lambda}\mu\tilde{\mu}$-calculus. In this section we prove this by showing a strong operational correspondence between the machine $M2$ and the machine $M1$.

The machine $M1$ and $M2$ are essentially the same, except that the former makes use of a global environment while the latter makes use of local explicit substitutions where terms and contexts carry their own private environment.

The strong operational correspondence is thus proved by mapping the explicit substitutions of $M2$ into the global environment of $M1$, caring that name clashes are avoided. We denote such a mapping by $[\![\cdot]\!]^{M}$.

Theorem 2 (Strong Operational Correspondence between $M2$ and $M1$). $[\![\cdot]\!]^{M}$ *is a strong operational correspondence between $M2$ and $M1$, i.e for all s and s'*

$$s \;\to^{M2}\; s' \;\Leftrightarrow\; [\![s]\!]^{M} \;\to^{M1}\; [\![s']\!]^{M}$$

where states from $M2$ and $M1$ are considered equal up-to relations $\equiv^{M2}$ and $\equiv^{M1}$, respectively.

Theorem 2 makes the diagram of Figure 1 commute. The final result of Figure 1, which is the strong operational correspondence between $M2$ and images in π-calculus, follows by composition of Theorems 1 and 2.

7 An asynchronous encoding

In this section we provide an encoding of the $\bar{\lambda}\mu\tilde{\mu}$-calculus into the asynchronous π-calculus $A\pi$ [9, 2]. Such an encoding is necessary in order to provide a distributed implementation of the calculus. We employ a variant of the polyadic $A\pi$-calculus able to perform both match and mismatch of channel names. The mapping $[\![\cdot]\!]^{a}$ is defined below.

$$\llbracket \langle v \mid e \rangle \rrbracket^a = \nu v, e \, (\llbracket v \rrbracket_v^a \mid \llbracket e \rrbracket_e^a \mid arbiter(v, e))$$

$$\llbracket \mu\beta. c \rrbracket_v^a = \nu x, y \, (\overline{v}\langle \mu, x, y \rangle \mid !x(v). \overline{v}\langle \mu, x, y \rangle \mid !y(\beta, \beta Test). \llbracket c \rrbracket^a)$$

$$\llbracket \tilde{\mu}x. c \rrbracket_e^a = \nu x, y \, (\overline{e}\langle \tilde{\mu}, x, y \rangle \mid !x(e). \overline{e}\langle \tilde{\mu}, x, y \rangle \mid !y(x, xTest). \llbracket c \rrbracket^a)$$

$$\llbracket \lambda x. v_1 \rrbracket_v^a = \nu x, y \, (\overline{v}\langle \lambda, x, y \rangle \mid !x(v). \overline{v}\langle \lambda, x, y \rangle \mid !y(e, \delta, \delta Test). \llbracket \tilde{\mu}x. \langle v_1 \mid \delta \rangle \rrbracket_e^a)$$

$$\llbracket v_2 \cdot e \rrbracket_e^a = \nu x, y \, (\overline{e}\langle cons, x, y \rangle \mid !x(e). \overline{e}\langle cons, x, y \rangle \mid !y(v, \delta). (\llbracket v_2 \rrbracket_v^a \mid !\delta(e). \llbracket e \rrbracket_e^a))$$

$$\llbracket z \rrbracket_v^a = \nu x \, (\overline{v}\langle var, x, z \rangle \mid \overline{v}\langle xTest \rangle \mid !x(v). \overline{v}\langle var, x, z \rangle)$$

$$\llbracket \beta \rrbracket_e^a = \nu x \, (\overline{e}\langle var, x, \beta \rangle \mid \overline{e}\langle \beta Test \rangle \mid !x(v). \overline{e}\langle var, x, \beta \rangle)$$

$$arbiter(v, e) = v(type_v, x_v, y_v). e(type_e, x_e, y_e).$$
$$[type_v = \mu]. [type_e \neq var \ or \ \tilde{\mu}]. \overline{y_v}\langle x_e, yes \rangle \mid$$
$$[type_e = \tilde{\mu}]. [type_v \neq \mu \ or \ var]. \overline{y_e}\langle x_v, yes \rangle \mid$$
$$[type_v = \mu]. [type_e = \tilde{\mu}]. \nu a \, (\overline{a} \mid a. \overline{y_v}\langle x_e \rangle \mid a. \overline{y_e}\langle x_v \rangle) \mid$$
$$[type_v = var]. v(bound).$$
$$[bound = yes]. \nu new_v, new_e \, (\overline{y_v}\langle new_v \rangle \mid \overline{new_e}\langle type_e, x_e, y_e \rangle)$$
$$\mid \ arbiter(new_v, new_e)$$
$$[bound \neq yes]. \nu new_v, new_e \, (\overline{new_v}\langle varNotBound, x_v, y_v \rangle \mid$$
$$\overline{new_e}\langle type_e, x_e, y_e \rangle \mid arbiter(new_v, new_e))$$
$$[type_e = var]. [type_v \neq var]. v(bound).$$
$$[bound = yes]. \nu new_v, new_e \, (\overline{y_e}\langle new_e \rangle \mid \overline{new_v}\langle type_v, x_v, y_v \rangle$$
$$\mid \ arbiter(new_v, new_e))$$
$$[bound \neq yes]. \nu new_v, new_e \, (\overline{new_e}\langle varNotBound, x_e, y_e \rangle \mid$$
$$\overline{new_v}\langle type_v, x_v, y_v \rangle \mid arbiter(new_v, new_e))$$
$$[type_v = \lambda]. [type_e = cons].$$
$$\nu new_v, new_e, \delta \, (\overline{y_v}\langle new_e, \delta, yes \rangle \mid \overline{y_e}\langle new_v, \delta \rangle \mid arbiter(new_v, new_e))$$

The encoding of commands is the parallel execution of the term, the context and a third process that acts like an arbiter. Terms and contexts are indeed supposed not to interact each other directly as in the previous encoding, but to interact with an arbiter that mediates the communications. The encoding is parametrized by a channel name that is the channel the term and context use to communicate with the arbiter. Every term and context send to the arbiter three information:

- The type of the construct. This information is discriminated by the name of the channel. For examples μ-abstractions send the channel μ, and cons contexts send the channel *cons*. The full association is straightforward and not listed.
- The private channel x. This channel denotes where the term or context is located.
- The private channel y. The arbiter uses this channel to perform the task associated to the meaning of the term or context.

The arbiter, once taken these information, performs a matching over the names of channels in order to detect the type of the term and context involved. With this information it can discriminate the correct use of the channels xs and ys in order to perform the expected reduction.

The management of the variables is more involved. Every variable needs to carry the information about whether it is bound or it is not. For example the variable x recovers this information by means of the channel named $xTest$; if the variable is bound, the arbiter would set this channel to the distinguished channel yes, exactly dedicated to this purpose. Otherwise, the channel name remains $xTest$ and the arbiter can discriminate the two situations by means of the match and mismatch operator, i.e. whether $xTest$ is, or is not, equal to yes. To understand the reason of such a management, the reader may consider the variables treatment in the arbiter code; when an arbiter does know that a variable is linked to some term or context, it can safely perform an output to liberate a process. If the variable is free, then such an output would just stop the computation even in commands where it is not supposed to, as in $\langle y \mid \tilde{\mu}x.\,c \rangle$, with y free. When the arbiter does know that y is not bound, it iterates again the arbiter tagging the variable as *not bound*, so at least it will be available for capture the second time. Solutions to offer the desired behaviour in a single go using no mixed choices seems to necessarily lead to potential loops in iterating the arbiter.

In order to provide an asynchronous encoding we need to resolve somehow the mixed choice of the previous encoding. This is reflected by introducing an arbiter that mediates the communications between the two players who thus no longer interact directly with each other.

It is worth noting that by straightforward modifications we can bias the arbiter in order to always give the precedence to μ- or $\tilde{\mu}$-reductions when the critical pair occurs. The arbiter can indeed detect such a pair and acts consistently. Being able to resolve the critical pair allows to exploit Curien and Herbelin's two encodings of the λ-calculus into the $\bar{\lambda}\mu\tilde{\mu}$-calculus that yields respectively a call-by-value and a call-by-name reduction.

8   Conclusions and Future Works

We have provided the first encoding of the $\bar{\lambda}\mu\tilde{\mu}$-calculus in π-calculus and, following the ideas by Milner and the technique by Vasconcelos, we have extracted from the encoding and studied a reduction machine $M1$ based on a global environment. The reduction machine turns out to be operationally equivalent to (a variant of) another reduction machine $M2$ previously given for the same calculus.

The $\bar{\lambda}\mu\tilde{\mu}$-calculus, which in its typed version is Curry-Howard isomorphic to classical sequent calculus, has a number of peculiarities that make the study of its embedding into the π-calculus worthwhile: $\bar{\lambda}\mu\tilde{\mu}$-terms are close to machine states (being isomorphic to sequent calculus), and thus the encoding need not be parametric on the channels along which terms interact; terms (and contexts) are fully symmetric and interact in a peer-to-peer way, which makes it useful the use of synchronous features of the π-calculus such as mixed choice in the encoding.

We have provided an asynchronous encoding, which, as expected, needs to resolve the mixed choice and makes use of an arbiter. Such an encoding, albeit less efficient,

is easily customisable to tune the reduction strategy and can be a basis for distributed implementations based on asynchronous primitives.

Finally, the reduction in the $\bar{\lambda}\mu\tilde{\mu}$-calculus is strongly normalizing (being isomorphic to cut elimination for classical logic), therefore the encoding into π-calculus may help understanding termination in the π-calculus.

The closest work is due by Cardelli, van Bakel, and Vigliotti in [18], where the authors provide an encoding from the $\mathcal{X}$-calculus into π-calculus. The $\mathcal{X}$-calculus shares with the $\bar{\lambda}\mu\tilde{\mu}$-calculus similar roots; also the former is indeed Curry-Howard isomorphic to classical sequent calculus, but the details of the two calculi are significantly different.

A main objective of future work is the study of the expressiveness of the strongly normalizing π-fragment identified. Related to this, we would like to study whether the techniques for termination in the $\bar{\lambda}\mu\tilde{\mu}$-calculus can be transported onto the π-calculus so to to be able to prove termination properties for larger subsets of processes. In this direction, we first plan to study an extension of the $\bar{\lambda}\mu\tilde{\mu}$-calculus admitting several terms and contexts interacting all together. Whether the new calculus would retain strong normalization for typed terms and contexts appears to be a nontrivial problem.

Another line of future work is the study of typed version of the encodings. In the synchronous case, since three global channels are used to implement the three reduction of the calculus and since redexes of terms with different types are met during reduction, we expect the typing of the global π-channels to pose some difficulties. In the asynchronous case the scenario is even worse: a given channel may be used to accomplish different tasks and also to transmit and receive different numbers of arguments, depending on the particular term or context involved at runtime.

Finally, the symmetry of the calculus suggests that fusion-like calculi might also be interesting target calculi for the encoding.

References

1. Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, revised edition, 1984.
2. Gerard Boudol. Asynchrony and the pi-calculus. Technical Report RR-1702, INRIA, 1992.
3. Matteo Cimini, Claudio Sacerdoti Coen, and Davide Sangiorgi. Online appendix. http://nemendur.ru.is/matteo/appendixForFaPTaL.pdf.
4. 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.
5. Romain Demangeon, Daniel Hirschkoff, and Davide Sangiorgi. Mobile processes and termination. In Jens Palsberg, editor, *Semantics and Algebraic Specification*, volume 5700 of *Lecture Notes in Computer Science*, pages 250–273. Springer, 2009.
6. Yuxin Deng and Davide Sangiorgi. Ensuring termination by typability. *Inf. Comput.*, 204(7):1045–1082, 2006.
7. Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1989.
8. Hugo Herbelin. A lambda-calculus structure isomorphic to Gentzen-style sequent calculus structure. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 1995.

9. Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP*, pages 133–147. Springer-Verlag, 1991.

10. Robin Milner. Functions as processes. In *Proceedings of the seventeenth international colloquium on Automata, languages and programming*, pages 167–180, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

11. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part I. *Information and Computation (I&C)*, 100(1):1–40, 1992. An earlier version of this paper appeared as Technical Report ECS-LFCS-89-85 of University of Edinburgh, 1989.

12. Michel Parigot. Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction. In *Logic Programming and Automated Reasoning: International Conference LPAR '92 Proceedings, St. Petersburg, Russia*, pages 190–201. Springer-Verlag, 1992.

13. Davide Sangiorgi. Internal mobility and agent passing calculi. In *Proc. ICALP'95*, volume 944 of *Lecture Notes in Computer Science*, pages 672–684. Springer, 1995.

14. Davide Sangiorgi. From lambda to pi; or, rediscovering continuations. *Mathematical Structures in Computer Science*, 9(4):367–401, 1999.

15. Davide Sangiorgi. Termination of processes. *Mathematical Structures in Computer Science*, 16(1):1–39, 2006.

16. Davide Sangiorgi and David Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

17. Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard Isomorphism, Volume 149 (Studies in Logic and the Foundations of Mathematics)*. Elsevier Science Inc., New York, NY, USA, 2006.

18. Stephen van Bakel, Luca Cardelli, and Maria Grazia Vigliotti. From X to Pi: Representing Classical Sequent Calculus in Pi-calculus. In *International Workshop on Classical Logic and Computation (CLC'08)*, 2009.

19. Vasco T. Vasconcelos. Lambda and pi calculi, cam and secd machines. *Journal of Functional Programming*, 15(1):101–127, 2005.

20. Nobuko Yoshida, Martin Berger, and Kohei Honda. Strong normalisation in the π-Calculus. In *16th Annual IEEE Symposium on Logic in Computer Science (LICS'01)*, pages 311–322. IEEE Computer Society, 2001.