HAL open science

An Output-Based Semantics of $\Lambda\mu$ with Explicit Substitution in the π-Calculus

Steffen Van Bakel, Maria Grazia Vigliotti

▶ **To cite this version:**

Steffen Van Bakel, Maria Grazia Vigliotti. An Output-Based Semantics of $\Lambda\mu$ with Explicit Substitution in the π-Calculus. 7th International Conference on Theoretical Computer Science (TCS), Sep 2012, Amsterdam, Netherlands. pp.372-387, 10.1007/978-3-642-33475-7_26 . hal-01556215

HAL Id: hal-01556215

https://inria.hal.science/hal-01556215v1

Submitted on 4 Jul 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

An output-based semantics of $\Lambda\mu$ with explicit substitution in the π-calculus

Extended Abstract

Steffen van Bakel and Maria Grazia Vigliotti

Department of Computing, Imperial College London, 180 Queen's Gate, London SW7 2BZ, UK
s.vanbakel@imperial.ac.uk, maria.vigliotti@imperial.ac.uk

Abstract. We study the $\Lambda\mu$-calculus, extended with explicit substitution, and define a compositional *output*-based translation into a variant of the π-calculus with pairing. We show that this translation preserves single-step explicit head reduction with respect to contextual equivalence. We use this result to show operational soundness for head reduction, adequacy, and operational completeness. Using a notion of implicative type-context assignment for the π-calculus, we also show that assignable types are preserved by the translation. We finish by showing that termination is preserved.

Introduction

Over the last two decades, the π-calculus [24] and its dialects have proven to give an interesting and expressive model of computation. Encodings of variants of the pure λ-calculus [14, 11] started with [24], which quickly led to more thorough investigations in [29, 31, 10] and also in the direction of object oriented calculi [21, 31].

For these encodings, over the years strong properties have been shown like soundness, completeness, termination, and full abstraction. The strength of these results has encouraged the investigation of encodings into the π-calculus of calculi that have their foundation in classical logic, as done in, for example, [22, 8, 15]. From these papers it might seem that the encoding of such calculi comes at a great price; for example, to encode typed $\lambda\mu$ [25], [22] needs to consider a version of the π-calculus that is not only strongly typed, but, moreover, allows reduction under guard and under replication; [8] shows preservation of reduction in $\mathcal{X}$ [9] only with respect to $\sqsubseteq_{\mathsf{c}}$, the contextual ordering; [15] defines a non-compositional encoding of $\overline{\lambda}\mu\tilde{\mu}$ [17] that strongly depends on recursion, and does not regard the logical aspect at all.

In this paper, we will show that it *is* possible to define a intuitive, natural, logical encoding of $\lambda\mu$ into the pure π-calculus that satisfies all the good properties. Although one could justifiably argue that calculi like $\mathcal{X}$ and $\overline{\lambda}\mu\tilde{\mu}$ are more expressive and, through their direct link to Gentzen's LK [18], more elegantly deal with negation and classical logic, they are also both symmetric in nature, which makes an accurate treatment in the π-calculus more intricate, as can be observed in [8, 15]. Moreover, as argued in [6, 5, 7], only for $\lambda\mu$ is it possible to define a filter semantics, which seems to strengthen the case for that calculus even more.

Reduction in $\lambda\mu$ is confluent and non-symmetric; in fact, the main reduction rule (and the only cause for non-termination, for example) is the β-reduction rule of the λ-calculus. In addition to that rule, $\lambda\mu$ has *structural* rules, where elimination takes place for a type that is not the type of the term itself, but rather for one that appears in one of the alternative conclusions of the shape $\alpha{:}A$, where the Greek variable is the name given to a sub-term. For the naming feature, $\lambda\mu$ adds $[\alpha]M$ to the syntax which expresses that α serves as a pointer to the term M, and pairs this with a notion of μ-abstraction $\mu\alpha.M$, which is used to redirect operands (terms) to those called α. It is this naming feature, together with the structural rules, that make $\lambda\mu$ difficult to reason over; this is reflected in [20] and [9], where the encoding of $\lambda\mu$ into $\overline{\lambda}\mu\tilde{\mu}$ and $\mathcal{X}$, respectively, does not respect normal reduction. In contrast, through our translation we will show that it is possible to give a process semantics for $\lambda\mu$ that very clearly shows that the *context switch* $\mu\alpha.[\beta]M$ is, essentially, just a variant of application.

For the construction of our translation, we will start with that defined in [10], that interprets terms under *output* rather than under *input*, by giving a name to the anonymous output of λ-terms; we will combine this with the inherent naming mechanism of $\lambda\mu$. To accurately define the notion of reduction that is modelled by our translation, we will define *untyped* $\Lambda\mu\mathbf{x}$, a version with explicit substitution [1, 12] of the $\Lambda\mu$-calculus [19], itself a variant of $\lambda\mu$, together with a notion of *explicit head reduction*[1], where reduction is also allowed under abstraction. We will define a new compositional semantic translation of $\Lambda\mu\mathbf{x}$ into the π-calculus, and show that it fully respects each individual explicit head reduction step.

Perhaps surprisingly, we do not need to extend the kind of process calculus at all to accomodate our translation, but can build that directly on the standard π-calculus; in particular, the naming and μ-binding features of $\lambda\mu$ are dealt with by the naming feature of the translation, and renaming, respectively. The only noteworthy change is that, when representing application MN, the communication needs to be replicated; the translation of application and structural substitution is almost identical.

The advantage of considering explicit substitution rather than the standard implicit substitution as considered in [31] has been strongly argued in [10]. That paper showed that communication in the π-calculus has a fine semantic level of granularity that 'faithfully mimics' explicit substitution, and not the implicit one; we stress this point again with the results presented in this paper.

1 The $\Lambda\mu$ calculus

The $\lambda\mu$-calculus is a proof-term syntax for classical logic, expressed in Natural Deduction, defined as an extension of the Curry type assignment system for the λ-calculus; we focus on de Groote's $\Lambda\mu$, a variant that splits the naming from the μ-binding. We will define in particular $\Lambda\mu\mathbf{x}$, a variant of $\Lambda\mu$ with explicit substitution *à la* $\lambda\mathbf{x}$ [12], and show our results for $\Lambda\mu\mathbf{x}$; since $\Lambda\mu\mathbf{x}$ implements $\Lambda\mu$-reduction, this implies that we also show some of our results for normal reduction (with implicit substitution).

[1] Called *spine reduction* in [10], and *head spine-reduction* in [32]; we prefer to use the terminology *head reduction* from [33].

Definition 1 (Syntax of $\Lambda\mu$). The $\Lambda\mu$-*terms* we consider are defined over the set of *term variables* represented by Roman characters, and *names*, or *context* variables, represented by Greek characters, through the grammar:

$$\begin{array}{lcccccccccc} M,N & ::= & x & | & \lambda x.M & | & MN & | & \mu\alpha.M & | & [\beta]M \\ & & \textit{variable} & & \textit{abstraction} & & \textit{application} & & \textit{context abstraction} & & \textit{naming} \end{array}$$

The notion of free and bound names is defined as can be expected, taking both λ and μ as binders, and we assume Barendregt's convention.

Simple type assignment for $\Lambda\mu$ is defined as follows:

Definition 2 (Types, Contexts, and Typing).

1. Types are defined by:
$$A,B \ ::= \ \varphi \mid \bot \mid A{\to}B \quad (A \neq \bot)$$
where φ is a basic type of which there are infinitely many.
2. A *context of inputs* Γ is a mapping from term variables to types, denoted as a finite set of *statements* $x{:}A$, such that the *subjects* of the statements (x) are distinct. We write Γ_1, Γ_2 for the *compatible* union of Γ_1 and Γ_2 (if $x{:}A_1 \in \Gamma_1$ and $x{:}A_2 \in \Gamma_2$, then $A_1 = A_2$), and write $\Gamma, x{:}A$ for $\Gamma, \{x{:}A\}$.
3. Contexts of *outputs* Δ, and the notions Δ_1, Δ_2 and $\alpha{:}A, \Delta$ are defined similarly.
4. Type assignment for $\Lambda\mu$ is defined by the following natural deduction system.

$$(Ax): \ \frac{}{\Gamma, x{:}A \vdash x : A \mid \Delta} \quad (\mu): \ \frac{\Gamma \vdash M : \bot \mid \alpha{:}A, \Delta}{\Gamma \vdash \mu\alpha.M : A \mid \Delta} \quad (\bot): \ \frac{\Gamma \vdash M : A \mid \beta{:}A, \Delta}{\Gamma \vdash [\beta]M : \bot \mid \beta{:}A, \Delta}$$

$$(\to I): \ \frac{\Gamma, x{:}A \vdash M : B \mid \Delta}{\Gamma \vdash \lambda x.M : A{\to}B \mid \Delta} \quad (\to E): \ \frac{\Gamma \vdash M : A{\to}B \mid \Delta \quad \Gamma \vdash N : A \mid \Delta}{\Gamma \vdash MN : B \mid \Delta}$$

In $\Lambda\mu$, reduction of terms is expressed via implicit substitution; as usual, $M[N/x]$ stands for the substitution of all occurrences of x in M by N, and $M[N{\cdot}\gamma/\alpha]$, the *structural substitution*, stands for the term obtained from M in which every sub-term of the form $[\alpha]M'$ is replaced by $[\gamma](M'N)$.

We have the following rules of computation in $\lambda\mu$:

Definition 3 ($\Lambda\mu$ reduction). $\Lambda\mu$ has two *computational rules*:

$$\begin{array}{rrcl} \textit{logical}\ (\beta): & (\lambda x.M)N & \to & M[N/x] \\ \textit{structural}\ (\mu): & (\mu\alpha.M)N & \to & \mu\gamma.M[N{\cdot}\gamma/\alpha] \quad \gamma\ \textit{fresh} \end{array}$$

as well as the *simplification rules*:

$$\begin{array}{rrcl} \textit{renaming}: & \mu\alpha.[\beta]\mu\gamma.M & \to & \mu\alpha.M[\beta/\gamma] \\ \textit{erasing}: & \mu\alpha.[\alpha]M & \to & M \quad \textit{if}\ \alpha\ \textit{does not occur in}\ M. \end{array}$$

(which are added mainly to simplify the presentation of results), and the contextual rules. We use $\to_{\beta\mu}$ for this reduction, and $\to^*_{\beta\mu}$ for its reflexive and transitive closure.

[26] has shown that typeable terms are strongly normalisable. It also defines extensional rules, that we do not consider here: the model we present through our translation is not extensional, and we can therefore not show that those rules are preserved by the translation. That this notion of reduction is confluent was shown in [28].

2 The synchronous π-calculus with pairing

The notion of π-calculus that we consider in this paper is similar to the one used also in [2], and is different from other systems studied in the literature [21] in that it adds pairing, and uses a *let*-construct to deal with inputs of pairs of names that get distributed.

As already argued in [10], the main reason for the addition of pairing [2] lies in preservation of (implicate, or functional) type assignment; therefore *data* is introduced as a structure over names, such that not only names but also pairs of names can be sent.

Definition 4 (Processes). *Channel names* and *data* are defined by:

$$a, b, c, d, x, y, z \quad \textit{names} \qquad\qquad p \ ::= \ a \mid \langle a,b \rangle \quad \textit{data}$$

Notice that pairing is *not* recursive. Processes are defined by:

$$P, Q \ ::= \ 0 \mid P|Q \mid !P \mid (\nu a)\, P \mid a(x).P \mid \overline{a}\langle p \rangle .\, P \mid \mathit{let}\, \langle x,y \rangle = p \ \mathit{in}\ P$$

A *context* $\mathsf{C}[\cdot]$ is a process with a hole $[\,]$; we call $a(x)$ and $\overline{a}\langle p \rangle$ *guards*, and call P in $a(x).\, P$ and $\overline{a}\langle p \rangle .\, P$ a process *under guard*.

We abbreviate $a(x).\, \mathit{let}\, \langle y,z \rangle = x \ \mathit{in}\ P$ by $a(y,z).\, P$, as well as $(\nu m)\,(\nu n)\, P$ by $(\nu mn)\, P$, and write $\overline{a}\langle p \rangle$ for $\overline{a}\langle p \rangle .\, 0$, and $\overline{a}\langle c,d \rangle .P$ for $\overline{a}\langle\langle c,d \rangle\rangle .\, P$. Notice that $\mathit{let}\, \langle x,y \rangle = a \ \mathit{in}\ P$ (where a is not a variable) is *stuck*.

Definition 5 (Congruence). The structural congruence is the smallest equivalence relation closed under contexts defined by the following rules:

$$\begin{array}{rcl@{\qquad}rcl}
P \mid 0 & \equiv & P & (P \mid Q) \mid R & \equiv & P \mid (Q \mid R) \\
P \mid Q & \equiv & Q \mid P & (\nu n)\, 0 & \equiv & 0 \\
!P & \equiv & P \mid !P & (\nu m)\,(\nu n)\, P & \equiv & (\nu n)\,(\nu m)\, P \\
!P & \equiv & !P \mid !P & (\nu n)\,(P \mid Q) & \equiv & P \mid (\nu n)\, Q \quad \textit{if } n \notin \mathit{fn}(P) \\
 & & & \mathit{let}\, \langle x,y \rangle = \langle a,b \rangle \ \mathit{in}\ P & \equiv & P[a/x, b/y]
\end{array}$$

As usual, we will consider processes modulo congruence and modulo α-convergence: this implies that we will not deal explicitly with the process $\mathit{let}\, \langle x,y \rangle = \langle a,b \rangle \ \mathit{in}\ P$, but rather with $P[a/x, b/y]$. We write $a \rightarrowtriangle b$ for the *forwarder* [31] $a(x).\, \overline{b}\langle x \rangle$.

Computation in the π-calculus with pairing is expressed via the exchange of *data*.

Definition 6 (Reduction). The *reduction relation* over the processes of the π-calculus is defined by the following (elementary) rules:

$$\begin{array}{rcl}
\overline{a}\langle p \rangle .\, P \mid a(x).Q & \rightarrow_\pi & P \mid Q[p/x] \\
P \rightarrow_\pi P' & \Rightarrow & (\nu n)\, P \rightarrow_\pi (\nu n)\, P' \\
P \rightarrow_\pi P' & \Rightarrow & P \mid Q \rightarrow_\pi P' \mid Q \\
P \equiv Q \ \& \ Q \rightarrow_\pi Q' \ \& \ Q' \equiv P' & \Rightarrow & P \rightarrow_\pi P'
\end{array}$$

As usual, we write $\rightarrow_\pi^+$ for the transitive closure of $\rightarrow_\pi$, and $\rightarrow_\pi^*$ for its reflexive and transitive closure; we write $\rightarrow_\pi (a)$ if we want to point out that a synchronisation took place over channel a, and write $\rightarrow_\pi (=_\alpha)$ if we want to point out that α-conversion has taken place during the synchronisation.

Notice that $\overline{a}\langle b,c \rangle \mid a(x,y).Q \ \rightarrow_\pi \ Q[b/x, c/y]$.

Definition 7. *1. We write $P \downarrow n$ and say that P* outputs on *n (or P exhibits an output barb on n) if $P \equiv (\nu b_1) \ldots b_m(\overline{n}\langle p\rangle \mid Q)$ for some Q, where $n \neq b_1 \ldots b_m$.*

2. We write $P \Downarrow n$ (P may *output* on *n) if there exists Q such that $P \rightarrow_\pi^* Q$ and $Q \downarrow n$.*

3. We write $P \sim_{\mathrm{C}} Q$ (P and Q are contextually equivalent*) if, for all $\mathsf{C}[\cdot]$, and for all n, $\mathsf{C}[P] \Downarrow n$ if and only if $\mathsf{C}[Q] \Downarrow n$.*

4. We write $\sim_{\mathrm{G}}$ (called garbage collection*) when we ignore a process because it is contextually equivalent to 0; notice that $\sim_{\mathrm{G}} \subset \sim_{\mathrm{C}}$.*

The following is a well-known result.

Proposition 8. Let P, Q not contain a, then

$$\begin{array}{lcl} (\nu a)\,(\overline{a}\langle b\rangle.\,P \mid a(x).\,Q) & \sim_{\mathrm{C}} & P \mid Q[b/x] \\ (\nu a)\,(!\overline{a}\langle b\rangle.\,P \mid a(x).\,Q) & \sim_{\mathrm{C}}, \sim_{\mathrm{G}} & Q[b/x] \end{array}$$

The π-calculus is equipped with a rich type theory [31], from the basic type system for counting the arity of channels [27] to sophisticated linear types in [22]. The notion of type assignment we use here is the one first defined in [8] and differs from systems presented in the past in that types do not contain channel information, and in that it expresses *implication*, *i.e.* has functional types and describes the '*input-output interface*' of a process.

Definition 9 (Context assignment for π [8]). Functional type assignment for the π-calculus is defined by the following sequent system:

$$(0):\ \frac{}{0:\Gamma\vdash\Delta} \qquad (!):\ \frac{P:\Gamma\vdash\Delta}{!\,P:\Gamma\vdash\Delta} \qquad (in):\ \frac{P:\Gamma,x{:}A\vdash x{:}A,\Delta}{a(x).P:\Gamma,a{:}A\vdash\Delta}$$

$$(\nu):\ \frac{P:\Gamma,a{:}A\vdash a{:}A,\Delta}{(\nu a)\,P:\Gamma\vdash\Delta} \qquad (out):\ \frac{P:\Gamma,b{:}A\vdash b{:}A,\Delta}{\overline{a}\langle b\rangle.\,P:\Gamma,b{:}A\vdash a{:}A,b{:}A,\Delta}\ (a\neq b)$$

$$(|):\ \frac{P:\Gamma\vdash\Delta \quad Q:\Gamma\vdash\Delta}{P\mid Q:\Gamma\vdash\Delta} \qquad (\textit{pair-out}):\ \frac{P:\Gamma,b{:}A\vdash c{:}B,\Delta}{\overline{a}\langle b,c\rangle.P:\Gamma,b{:}A\vdash a{:}A\rightarrow B,c{:}B,\Delta}\ (b\notin\Delta; a,c\notin\Gamma)$$

$$(W):\ \frac{P:\Gamma\vdash\Delta}{P:\Gamma'\vdash\Delta'}\ (\Gamma'\supseteq\Gamma,\Delta'\supseteq\Delta) \qquad (\textit{let}):\ \frac{P:\Gamma,y{:}B\vdash x{:}A,\Delta}{\textit{let}\,\langle x,y\rangle = z\ \textit{in}\ P:\Gamma,z{:}A\rightarrow B\vdash\Delta}\ (y,z\notin\Delta; x\notin\Gamma)$$

We adjust the system for the type constant $\bot$ by allowing that only in right-hand contexts. We write $P:\Gamma\vdash_\pi\Delta$ if there exists a derivation using these rules that has this expression in the conclusion.

We should perhaps stress that it is not known if this system has a relation with logic.

The following rule is derivable:

$$(\textit{pair-in}):\ \frac{P:\Gamma,y{:}B\vdash_\pi x{:}A,\Delta}{a(x,y).\,P:\Gamma,a{:}A\rightarrow B\vdash_\pi\Delta}\ (y,a\notin\Delta,\ x\notin\Gamma)$$

The soundness result is stated as:

Theorem 10 (Witness reduction [8]). If $P:\Gamma\vdash_\pi\Delta$ and $P\rightarrow_\pi Q$, then $Q:\Gamma\vdash_\pi\Delta$.

3 Context and background of this paper

In the past, there have been several investigations of encoding from the λ-calculus [11] into the π-calculus [24, 29]. Research in this direction started by Milner's encoding $[\![\cdot]\!]^{\mathrm{M}}\cdot$ of λ-terms [24]; Milner's encoding is *input* based and the translation of closed λ-terms respects large-step *lazy* reduction $\rightarrow_{\mathrm{L}}$ [3] to normal form up to substitution. Standard operational soundness result hold for this translation, and full abstraction has been shown by in [29] for an (input-based, as Milner's) encoding $\mathcal{H}^{\llbracket\cdot\rrbracket}\langle\cdot\rangle$, of the lazy λ-calculus into the higher-order π-calculus (where in synchronisation not names are sent, but processes).

In [10], we presented a *logical*, *output-based* translation $\llbracket\cdot\rrbracket^{\mathrm{S}}\cdot$ that interprets abstraction $\lambda x.M$ not using *input*, but via an asynchronous *output* which leaves the translation of the body M free to reduce. That translation is defined as:

$$\begin{array}{rcl}
\llbracket x\rrbracket^{\mathrm{S}}a & \triangleq & x(w).\overline{a}\langle w\rangle \\
\llbracket\lambda x.M\rrbracket^{\mathrm{S}}a & \triangleq & (\nu xb)\,(\llbracket M\rrbracket^{\mathrm{H}}b \mid \overline{a}\langle x,b\rangle) \\
\llbracket MN\rrbracket^{\mathrm{S}}a & \triangleq & (\nu c)\,(\llbracket M\rrbracket^{\mathrm{H}}c \mid c(v,d).\,(!\llbracket N\rrbracket^{\mathrm{H}}v \mid d\rightarrow a)) \\
\llbracket M\langle x:=N\rangle\rrbracket^{\mathrm{S}}a & \triangleq & (\nu x)\,(\llbracket M\rrbracket^{\mathrm{H}}a \mid !\llbracket N\rrbracket^{\mathrm{H}}x)
\end{array}$$

For this translation, [10] showed (using $\uparrow$ to denote non-termination)

1. $M\uparrow \Rightarrow \llbracket M\rrbracket^{\mathrm{S}}a\uparrow$, and $M \rightarrow_{\mathbf{xH}} N \Rightarrow \llbracket M\rrbracket^{\mathrm{S}}a \rightarrow_{\pi}^{*}\sim_{\mathrm{C}} \llbracket N\rrbracket^{\mathrm{H}}a$.
2. $\Gamma \vdash M : A \Rightarrow \llbracket M\rrbracket^{\mathrm{S}}a : \Gamma \vdash_{\pi} a{:}A$.

As argued in [10], to show the above result, which formulates a direct *step-by-step* relation between β-reduction and the synchronisation in the π-calculus, it is necessary to make the substitution explicit. This is a direct result of the fact that, in the π-calculus, λ's implicit substitution gets 'implemented' *one variable at the time*, rather than all in one fell swoop. Since we aim to show a similar result for $\Lambda\mu$, we will therefore define a notion of explicit substitution. Although termination is not studied in that paper, it is easily achieved through restricting the notion of reduction in the π-calculus by not allowing reduction to take place inside processes whose output cannot be received, or by placing a guard on the replication as we do in this paper.

A natural extension of this line of research is to see if the π-calculus can be used to interpret more complex calculi as well, as for example calculi that relate not to intuitionistic logic, but to classical logic, as $\lambda\mu$, $\overline{\lambda}\mu\tilde{\mu}$, or $\mathcal{X}$. There are, to date, a number of papers on this topic. In [22] an interpretation of Call-by-Value $\lambda\mu$ is defined that is based on Milner's. The authors consider *typed processes only*, and use a much more liberal notion of reduction on processes by allowing reduction *under* guards, making the resulting calculus very different from the original π-calculus. Types for processes prescribe usage of names

In [8] an interpretation into π of the sequent calculus $\mathcal{X}$ is defined that enjoys the Curry-Howard isomorphism for Gentzen's LK [18], which is shown to respect reduction. However, this result is only partial, as it is formulated as "*if* $P \rightarrow_{\mathcal{X}} Q$*, then* $[\![P]\!] \,{}_{\mathrm{C}}\!\sqsupseteq [\![Q]\!]$", allowing $[\![P]\!]$ to have more observable behaviour than $[\![Q]\!]$. Although in [8] it is reasoned that this is natural in the context of non-confluent, symmetric sequent calculi, and is shown that the interpretation preserves types, it is a weaker result than could perhaps be expected.

An encoding of $\overline{\lambda}\mu\tilde{\mu}$ is studied in [15]; the interpretation defined there strongly depends on recursion, is not compositional, and preserves only outermost reduction; no relation with types is shown.

4 $\Lambda\mu$ with explicit substitution

One of the main achievements of [10] is that it establishes a strong link between reduction in the π-calculus and step-by-step *explicit substitution* for the λ-calculus, by formulating a result not only with respect to explicit head reduction and the spine encoding defined there, but also for Milner's encoding with respect to explicit lazy reduction.

In view of this, we decided to study a variant of $\Lambda\mu$ with explicit substitution as well, and present here $\Lambda\mu\mathbf{x}$. Explicit substitution treats substitution as a first-class operator, both for the logical and the structural substitution, and describes all the necessary steps to effectuate both.

Definition 11 ($\Lambda\mu\mathbf{x}$). 1. The syntax of the *explicit* $\Lambda\mu$ *calculus*, $\Lambda\mu\mathbf{x}$, is defined by:

$$M, N \ ::= \ x \mid \lambda x.M \mid MN \mid M\langle x := N\rangle \mid \mu\alpha.M \mid [\beta]M \mid M\langle \alpha := N\cdot\gamma\rangle$$

We call a term *pure* if it does not contain explicit substitution.

2. The reduction relation $\to_{\mathbf{x}}$ on terms in $\Lambda\mu\mathbf{x}$ is defined as the compatible closure of the rules (we only show the important ones):
   (a) Main reduction rules:

$$\begin{array}{rcll} (\lambda x.M)N & \to & M\langle x := N\rangle & N \textit{ pure} \\ (\mu\alpha.M)N & \to & \mu\gamma.M\langle \alpha := N\cdot\gamma\rangle & N \textit{ pure} \\ \mu\beta.[\beta]M & \to & M & \textit{if } \beta \notin \mathit{fn}(M) \\ \mu\beta.[\delta]\mu\gamma.M & \to & \mu\beta.M[\delta/\gamma] & \end{array}$$

   (b) Term substitution rules, like

$$\begin{array}{rcll} x\langle x := N\rangle & \to & N & \\ M\langle x := N\rangle & \to & M & x \notin \mathit{fv}(M) \end{array}$$

   (c) Structural rules, like

$$\begin{array}{rcll} ([\alpha]M)\langle \alpha := N\cdot\gamma\rangle & \to & [\gamma](M\langle \alpha := N\cdot\gamma\rangle)N & \\ ([\beta]M)\langle \alpha := N\cdot\gamma\rangle & \to & [\beta](M\langle \alpha := N\cdot\gamma\rangle) & \alpha \neq \beta \\ M\langle \alpha := N\cdot\gamma\rangle & \to & M & \alpha \notin \mathit{fn}(M) \end{array}$$

   (d) Contextual rules, like

$$M \to N \ \Rightarrow \ \begin{cases} ML & \to \ NL \\ LM & \to \ LN \\ M\langle x := L\rangle & \to \ N\langle x := L\rangle \\ L\langle \alpha := M\cdot\gamma\rangle & \to \ L\langle \alpha := N\cdot\gamma\rangle \end{cases}$$

3. We define $\to_{:=}$ as the notion of reduction where the main reduction rules are not used, and $=_{\mathbf{x}}$ as the smallest equivalence relation generated by $\to_{\mathbf{x}}$.

Notice that this is a system different from that of [4], where a version with explicit substitution is defined for a variant of $\lambda\mu$ that uses de Bruijn indices [13].

Explicit substitution describes explicitly the process of executing a $\beta\mu$-reduction, *i.e.* expresses syntactically the details of the computation as a succession of atomic steps (like in a first-order rewriting system), where the implicit substitution of each $\beta\mu$-reduction step is split up into reduction steps. Thereby the following is straightforward:

Proposition 12 ($\Lambda\mu\mathbf{x}$ implements $\Lambda\mu$-reduction). 1. $M \to_{\beta\mu} N \Rightarrow M \to_{\mathbf{x}}^{*} N$.
2. $M \in \Lambda\mu$ & $M \to_{\mathbf{x}} N \Rightarrow \exists\, L \in \Lambda\mu\ [N \to_{:=}^{*} L]$.

The notion of type assignment on $\Lambda\mu\mathbf{x}$ is a natural extension of the system for the $\Lambda\mu$-calculus of Def. 2 by adding rules $(T\text{-}cut)$ and $(C\text{-}cut)$.

Definition 13. *Using the notion of type assignment in Def. 2, type assignment for $\Lambda\mu\mathbf{x}$ is defined by adding:*

$$(T\text{-}cut):\ \frac{\Gamma, x{:}A \vdash M : B \mid \Delta \quad \Gamma \vdash N : A \mid \Delta}{\Gamma \vdash M\langle x := N\rangle : B \mid \Delta}$$

$$(C\text{-}cut):\ \frac{\Gamma \vdash M : C \mid \alpha{:}A{\to}B, \gamma{:}B, \Delta \quad \Gamma \vdash N : A \mid \gamma{:}B, \Delta}{\Gamma \vdash M\langle \alpha := N{\cdot}\gamma\rangle : C \mid \Delta}$$

We write $\Gamma \vdash_{\mu\mathbf{x}} M : A$ for judgements derivable in this system.

We also consider the notion of head reduction;

Definition 14. *1. We define* head *reduction $\to_{\mathrm{H}}$ as a restriction of $\to_{\beta\mu}$ by removing the contextual rule $M \to N \ \Rightarrow\ LM \to LN$.*
2. The $\Lambda\mu$ and $\Lambda\mu\mathbf{x}$ head-normal forms are defined through the grammar:

$$\begin{array}{rcl} \boldsymbol{H} & ::= & xM_1 \cdots M_n\ (n \geq 0) \mid \lambda x.\boldsymbol{H} \mid [\alpha]\boldsymbol{H} \\ & \mid & \mu\alpha.\boldsymbol{H}\ \ (\boldsymbol{H} \neq [\alpha]\boldsymbol{H}' \ \&\ \alpha \notin \boldsymbol{H}', \boldsymbol{H} \neq [\beta]\gamma.\boldsymbol{H}') \end{array}$$

3. The head variable *of M, $hv(M)$, and* head name *$hn(M)$ are defined as expected.*

The following is straightforward:

Proposition 15 ($\to_{\mathrm{H}}$ implements $\Lambda\mu$'s head reduction). If $M \to_{\beta\mu}^{*} N$ with N in head-normal form, then there exists L in $\to_{\mathrm{H}}$-normal form such that $M \to_{\mathrm{H}}^{*} L$, and $L \to_{\beta\mu}^{*} N$, and none of these last steps are reductions in $\to_{\mathrm{H}}$.

Notice that $\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)) \ \to_{\mathrm{H}}\ \lambda f.f((\lambda x.f(xx))(\lambda x.f(xx)))$ and this last term is in head-normal form, and in $\to_{\mathrm{H}}$-normal form.

In the context of head reduction, we can economise further on how substitution is executed, and perform only those replacements of variables by terms that are essential for the continuation of reduction. We will therefore limit substitution to allow it to only replace the head variable or name of a term. We will show that this is exactly the kind of reduction that the π-calculus naturally encodes.

Definition 16 (Explicit head reduction cf. [10]). We define *explicit head reduction* $\to_{\mathbf{x}\mathrm{H}}$ on $\Lambda\mu\mathbf{x}$ as $\to_{\mathbf{x}}$, but for:

1. To avoid looping unnecessarily, application of all term substitution (resp. structural) rules on $M\langle x := N\rangle$ (resp. $M\langle \alpha := N{\cdot}\gamma\rangle$) is only allowed if $hv(M) = x$ (resp. $hn(M) = \alpha$); the only exception are the garbage collection rules, *i.e.* when $x \notin fv(M)$ ($\alpha \notin fn(M)$).

2. We change two cases:
$$\begin{array}{lcll} (PQ)\langle x := N\rangle & \to & (P\langle x := N\rangle\, Q)\langle x := N\rangle & (x = hv(P)) \\ (PQ)\langle \alpha := N\cdot\gamma\rangle & \to & (P\langle \alpha := N\cdot\gamma\rangle\, Q)\langle \alpha := N\cdot\gamma\rangle & (\alpha = hn(P)) \end{array}$$
3. We add two substitution rules:
$$\begin{array}{rcll} M\langle x := N\rangle\langle y := L\rangle & \to & M\langle y := L\rangle\langle x := N\rangle\langle y := L\rangle & (y = hv(M)) \\ M\langle \alpha := N\cdot\gamma\rangle\langle \beta := L\cdot\delta\rangle & \to & M\langle \beta := L\cdot\delta\rangle\langle \alpha := N\cdot\gamma\rangle\langle \beta := L\cdot\delta\rangle & (\alpha = hn(P)) \end{array}$$
4. We remove the contextual rules:
$$M \to N \;\Rightarrow\; \begin{cases} LM & \to\; LN \\ L\langle x := M\rangle & \to\; L\langle x := N\rangle \\ L\langle \alpha := M\cdot\gamma\rangle & \to\; L\langle \alpha := N\cdot\gamma\rangle \end{cases}$$

Notice that, for example, in case 2, the first of the two clauses postpones the substitution $\langle x := N\rangle$ on Q until such time that an occurrence of the variable x in Q becomes the head-variable. It is straightforward to show that this notion of reduction is confluent; remember that in $M\langle x := N\rangle$ and $M\langle \alpha := N\cdot\gamma\rangle$, N is a pure term.

The following proposition states the relation between explicit head reduction, head reduction, and explicit reduction.

Proposition 17. 1. If $M \to_{\mathrm{H}}^{*} N$, then there exists L such that $M \to_{\mathbf{x}\mathrm{H}}^{*} L$ and $N \to_{:=}^{*} L$.
2. If $M \to_{\mathrm{H}}^{*} N$ and N is in $\to_{\mathrm{H}}$-normal form, then there exists L such that $M \to_{\mathbf{x}\mathrm{H}}^{*} L$ and $N \to_{\mathbf{x}}^{*} L$.
3. If $M \to_{\mathbf{x}\mathrm{H}}^{*} N$ with $M \in \Lambda\mu$ and N is in $\to_{\mathbf{x}\mathrm{H}}$-normal form, then there exists $L \in \Lambda\mu$ such that $N \to_{:=} L$, and L is in $\Lambda\mu$ head-normal form.

This result gives that we can show our main results for $\Lambda\mu\mathbf{x}$ for reductions that reduce to head-normal form, that are naturally defined as follows:

Definition 18 (cf. [23]). The normal forms with respect to $\to_{\mathbf{x}\mathrm{H}}$ are defined through:
$$\begin{array}{rcll} \boldsymbol{N} & ::= & xM_1\cdots M_n\ (n \geq 0) \mid \lambda x.\boldsymbol{N} \mid [\alpha]\boldsymbol{N} & \\ & \mid & \mu\alpha.\boldsymbol{N} & (\boldsymbol{N} \neq [\alpha]\boldsymbol{N}' \;\&\; \alpha \notin \boldsymbol{N}', \boldsymbol{N} \neq [\beta]\gamma.\boldsymbol{N}') \\ & \mid & \boldsymbol{N}\langle x := M\rangle & (hv(\boldsymbol{N}) \neq x) \\ & \mid & \boldsymbol{N}\langle \alpha := M\cdot\gamma\rangle & (hn(\boldsymbol{N}) \neq \alpha) \end{array}$$

Notice that, for example, under head reduction, any term of the shape $(\lambda x.P)Q$ in one of the M_i in $xM_1\cdots M_n$ is *not* considered a redex.

5 A logical translation of $\Lambda\mu$x to π

We will now define our logical, output-based translation $\llbracket\cdot\rrbracket\cdot$ of the $\Lambda\mu\mathbf{x}$-calculus into the π-calculus. The main idea behind the translation, as in [10], is to give a name to the anonymous output of terms; it combines this with the inherent naming mechanism of $\Lambda\mu$. In the definition below, for readability, we use the symbol $\bullet$ as a channel name to represent an output that cannot be received from.

Definition 19 (Logical translation of $\Lambda\mu$x terms). The translation of $\Lambda\mu\mathbf{x}$ terms into the π-calculus is defined in Fig. 1.

$$
\begin{array}{rcl}
\llbracket x \rrbracket a & \triangleq & x(u).\,!\,u \rightarrow a \\
\llbracket \lambda x.M \rrbracket a & \triangleq & (\nu x b)\,(\llbracket M \rrbracket b \mid \overline{a}\langle x,b\rangle) \\
\llbracket MN \rrbracket a & \triangleq & (\nu c)\,(\llbracket M \rrbracket c \mid !\,c(v,d).\,(\llbracket v := N \rrbracket \mid !\,d \rightarrow a)) \\
\llbracket M\,\langle x := N\rangle \rrbracket a & \triangleq & (\nu x)\,(\llbracket M \rrbracket a \mid \llbracket x := N \rrbracket) \\
\llbracket x := N \rrbracket & \triangleq & !\,(\nu w)\,(\overline{x}\langle w\rangle.\,\llbracket N \rrbracket w) \\
\llbracket \mu\gamma.M \rrbracket a & \triangleq & (\nu \bullet)\,((\llbracket M \rrbracket \bullet)[a/\gamma]) \\
\llbracket [\beta]M \rrbracket a & \triangleq & \llbracket M \rrbracket \beta \\
\llbracket M\,\langle \beta := N\cdot\gamma\rangle \rrbracket a & \triangleq & (\nu\beta)\,(\llbracket M \rrbracket a \mid \llbracket \beta := N\cdot\gamma \rrbracket) \\
\llbracket \alpha := N\cdot\gamma \rrbracket & \triangleq & !\,\alpha(v,d).\,(\llbracket v := N \rrbracket \mid !\,d \rightarrow \gamma)
\end{array}
$$

Fig. 1. The logical translation

We would like to stress that. although inspired by logic, our translation does not depend on types *at all*; in fact, we can treat untypeable terms as well, and can show that $\llbracket (\lambda x.xx)(\lambda x.xx) \rrbracket a$ (perhaps the prototype of a non-typeable term) runs to itself (this already holds for $\llbracket \cdot \rrbracket^{\mathrm{H}} \cdot$ of [10]).

Notice that, as is the case for Milner's translation and in contrast to the interpretation of [10], a guard is placed on the replicated terms. This is not only done with an eye on proving preservation of termination, but more importantly, to make sure that $(\nu x)\,(\llbracket x := N \rrbracket) \sim_{\mathrm{C}} \mathit{0}$: since a term can have named sub-terms, the translation will generate output not only for the term itself, but also for those named terms, so $(\nu x)\,(\llbracket x := N \rrbracket)$ *can have* observable behaviour, in contrast to [10], where this process *is* equivalent to $\mathit{0}$.

We could have avoided the implicit renaming in the case for μ-abstraction and defined $\llbracket \mu\gamma.M \rrbracket a = (\nu\bullet\gamma)\,(\llbracket M \rrbracket \bullet \mid !\,\gamma \rightarrow a)$, which is operationally (contextually) the same as $(\nu\bullet)\,((\llbracket M \rrbracket \bullet)[a/\gamma])$, but then we could not show that terms in head-normal form are translated to processes in normal form (Lem. 24). There is a strong relation between this encoding and the abstract machine defined in [16], but for the fact that that only represents lazy reduction.

Notice that $\llbracket \mu\gamma.[\beta]M \rrbracket a \triangleq (\nu\bullet)\,((\llbracket M \rrbracket \beta)[a/\gamma])$, so had we considered to just encode $\lambda\mu$, we could have defined

$$\llbracket \mu\gamma.[\beta]M \rrbracket a \triangleq (\nu\bullet)\,((\llbracket M \rrbracket \beta)[a/\gamma]) = \llbracket M[a/\gamma] \rrbracket \beta$$

so $\lambda\mu$'s binding-and-naming has no representation in π.

Moreover, notice the similarity between

$$
\begin{array}{r}
\llbracket MN \rrbracket a = (\nu c)\,(\llbracket M \rrbracket c \mid !\,c(v,d).\,(\llbracket v := N \rrbracket \mid !\,d \rightarrow a)) \\
\llbracket M\,\langle \beta := N\cdot\gamma\rangle \rrbracket a = (\nu\beta)\,(\llbracket M \rrbracket a \mid !\,\beta(v,d).\,(\llbracket v := N \rrbracket \mid !\,d \rightarrow \gamma))
\end{array}
$$

The first communicates N via the output channel c of M, whereas the second communicates with all the sub-terms that have β as its output name[2]. This very elegantly expresses exactly what the structural substitution does: it 'connects' arguments with the correct position in a term; it also allows us to write $(\nu c)\,(\llbracket M \rrbracket c \mid \llbracket c := N\cdot a \rrbracket)$ for $\llbracket MN \rrbracket a$. This stresses that the π-calculus constitutes a very powerful abstract machine indeed: although the notion of structural reduction in $\lambda\mu$ is very different from normal β-reduction, no special measures had to be taken in order to be able to express it; the

[2] A similar observation can be made for the encoding of $\lambda\mu$ in $\mathcal{X}$; see [9].

component of our encoding that deals with pure λ-terms is almost exactly that of [10] (ignoring for the moment that substitution is modelled using a guard, which affects also the interpretation of variables), but for the use of replication in the case for application. In fact, the distributive character of structural substitution is dealt with entirely by congruence; see also Ex. 23. As standard in the literature [30], we say that a name a occurs in the *output subject position* of a process P if $P \Downarrow a$.

Lemma 20. 1. Assume that a is only used for output R, Q. Then:
$$(\nu a)\,(!a(x).\,P \mid Q \mid R) \;\sim_{\mathrm{C}}\; (\nu a)\,(!a(x).\,P \mid Q) \mid (\nu b)\,(!b(x).\,P \mid R[b/a])$$
2. Assume that a is only used for input in R, Q. Then:
$$(\nu a)\,(!\overline{a}\langle p\rangle.\,P \mid Q \mid R) \;\sim_{\mathrm{C}}\; (\nu a)\,(!\overline{a}\langle p\rangle.\,P \mid Q) \mid (\nu b)\,(!\overline{b}\langle p\rangle.\,P \mid R[b/a])$$
3. $(\nu a)\,(\llbracket P \rrbracket a \mid !a(p).\,Q) \;\sim_{\mathrm{C}}\; (\nu a)\,((\nu b)\,(\llbracket P \rrbracket b \mid !b(p).\,Q) \mid !a(p).\,Q)$

To underline the significance of our results, notice that the translation is not trivial, since $\lambda y.y$ and $\lambda yz.y$ are interpreted by, respectively, the processes $(\nu yb)\,(y(u).\,!u{\rightarrow}b \mid \overline{a}\langle y,b\rangle)$ and $(\nu yb)\,((\nu zb)\,(y(u).\,!u{\rightarrow}b \mid \overline{b}\langle z,b\rangle) \mid \overline{a}\langle y,b\rangle)$, that differ under $\sim_{\mathrm{C}}$.

It is straightforward to show that typeability is preserved:

Theorem 21 (Type preservation). If $\Gamma \vdash_{\mu\mathbf{x}} M : A \mid \Delta$, then $\llbracket M \rrbracket a : \Gamma \vdash_{\pi} a{:}A, \Delta$.

PROOF. By induction on the structure of derivations in $\vdash_{\mu\mathbf{x}}$; we only show one case:

(*C-cut*) Then $M = P\langle \alpha := Q{\cdot}\gamma\rangle$ and we have both $\Gamma \vdash_{\mu\mathbf{x}} P : C \mid \alpha{:}A{\rightarrow}B, \Delta$ and $\Gamma \vdash_{\mu\mathbf{x}} Q : A \mid \gamma{:}B, \Delta$ for some B. By induction, there exist $\mathcal{D}_1 :: \llbracket P \rrbracket a : \Gamma \vdash_{\pi} a{:}C, \alpha{:}A{\rightarrow}B, \Delta$ and, since a is fresh, $\mathcal{D}_2 :: \llbracket Q \rrbracket w : \Gamma \vdash_{\pi} w{:}B, \Delta$, and we can construct

$$\dfrac{\dfrac{\mathcal{D}_1}{\llbracket P \rrbracket a : \Gamma \vdash a{:}B{\rightarrow}A, \Delta} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}_2}{\llbracket Q \rrbracket w : \Gamma \vdash w{:}B, \Delta}}{\overline{b}\langle w\rangle.\,\llbracket Q \rrbracket w : \Gamma \vdash b{:}B, w{:}B, \Delta}\,(out)}{(\nu w)\,(\overline{b}\langle w\rangle.\,\llbracket Q \rrbracket w) : \Gamma \vdash b{:}B, \Delta}\,(\nu)}{!\,(\nu w)\,(\overline{b}\langle w\rangle.\,\llbracket Q \rrbracket w) : \Gamma \vdash b{:}B, \Delta}\,(!) \quad \dfrac{\dfrac{\dfrac{\dfrac{}{0 : w{:}A \vdash w{:}A}\,(0)}{\overline{\gamma}\langle w\rangle : w{:}A \vdash \gamma{:}A, w{:}A}\,(out)}{d{\rightarrow}\gamma : d{:}A \vdash \gamma{:}A}\,(in)}{!d{\rightarrow}\gamma : d{:}A \vdash \gamma{:}A}\,(!)}{\llbracket b := Q \rrbracket \mid !d{\rightarrow}\gamma : \Gamma, d{:}A \vdash \gamma{:}A, b{:}B, \Delta}\,(|)}{\alpha(b,d).\,(\llbracket b := Q \rrbracket \mid !d{\rightarrow}\gamma) : \Gamma, \alpha{:}B{\rightarrow}A \vdash \gamma{:}A, \Delta}\,(pair\text{-}in)}{!\alpha(b,d).\,(\llbracket b := Q \rrbracket \mid !d{\rightarrow}\gamma) : \Gamma, \alpha{:}B{\rightarrow}A \vdash \gamma{:}A, \Delta}\,(!)}{\llbracket P \rrbracket a \mid !\alpha(b,d).\,(\llbracket b := Q \rrbracket \mid !d{\rightarrow}\gamma) : \Gamma, \alpha{:}B{\rightarrow}A \vdash \gamma{:}A, \Delta}\,(|)}{(\nu\alpha)\,(\llbracket P \rrbracket a \mid \llbracket \alpha := Q{\cdot}\gamma \rrbracket) : \Gamma \vdash \gamma{:}A, \Delta}\,(\nu)$$

and $(\nu\alpha)\,(\llbracket P \rrbracket a \mid \llbracket \alpha := Q{\cdot}\gamma \rrbracket) \;=\; \llbracket P\langle \alpha := Q{\cdot}\gamma\rangle \rrbracket a$. □

We will now show that our translation fully respects the explicit reduction $\rightarrow_{\mathbf{x}}$, modulo contextual equivalence, using renaming of output and garbage collection. Renaming is defined and justified via the following lemma.

Lemma 22 (Renaming lemma). 1. $(\nu a)\,(!a{\rightarrow}e \mid \llbracket M \rrbracket a) \;\sim_{\mathrm{C}}\; \llbracket M[e/a] \rrbracket e$.
2. $(\nu a)\,(!a{\rightarrow}e \mid \llbracket M \rrbracket b) \;\sim_{\mathrm{C}}\; \llbracket M[e/a] \rrbracket b$.

We will use $\sim_{\mathrm{R}}$ if we want to emphasise that two processes are equivalent just using renaming and write $\rightsquigarrow^*_{\pi}$ for the relation $\rightarrow^*_{\pi} \cup \sim_{\mathrm{G}} \cup \sim_{\mathrm{R}}$.

Using this lemma, we can show the following:

$$
\begin{array}{ll}
\llbracket(\lambda x.x)(\mu\alpha.[\alpha](\lambda q.q)(\mu\beta.[\alpha]\lambda y.y))\rrbracket\, a & \triangleq \\
(\nu c)\,((\nu xb)\,(\llbracket x\rrbracket\, b \mid \overline{c}\langle x,b\rangle) \mid !\,c(v,d).\,(\llbracket v{:=}\mu\alpha.[\alpha](\lambda q.q)(\mu\beta.[\alpha]\lambda y.y)\rrbracket \mid !d{\to}a)) & \to (c) \\
(\nu xb)\,(\llbracket x\rrbracket\, b \mid \llbracket x{:=}\mu\alpha.[\alpha](\lambda q.q)(\mu\beta.[\alpha]\lambda y.y)\rrbracket \mid !b{\to}a) \mid (\nu c)\,(!\,c(v,d).\cdots) & \equiv, \triangleq, \sim_{\mathrm{G}} \\
(\nu xb)\,(x(u).\,!\,u{\to}b \mid !\,(\nu w)\,(\overline{x}\langle w\rangle.\,\llbracket\mu\alpha.[\alpha](\lambda q.q)(\mu\beta.[\alpha]\lambda y.y)\rrbracket\, w) \mid !\,b{\to}a) & \to (x) \\
(\nu wb)\,(!\,w{\to}b \mid \llbracket\mu\alpha.[\alpha](\lambda q.q)(\mu\beta.[\alpha]\lambda y.y)\rrbracket\, w \mid !\,b{\to}a) \mid (\nu x)\,(!\,(\nu w)\,(\overline{x}\langle w\rangle.\cdots)) & \triangleq, \sim_{\mathrm{G}}, =_{\alpha} \\
(\nu\alpha b)\,(!\,\alpha{\to}b \mid (\nu\bullet)\,(\llbracket[\alpha](\lambda q.q)(\mu\beta.[\alpha]\lambda y.y)\rrbracket\,\bullet) \mid !\,b{\to}a) & \triangleq, \equiv \\
(\nu\alpha b)\,(!\,\alpha{\to}b \mid (\nu c)\,((\nu qb_1)\,(\llbracket q\rrbracket\, b_1 \mid \overline{c}\langle q,b_1\rangle) \mid & \\
\qquad !\,c(v,d).\,(\llbracket v{:=}\mu\beta.[\alpha]\lambda y.y\rrbracket \mid !d{\to}\alpha)) \mid !\,b{\to}a) & \to (c), \sim_{\mathrm{G}}, \triangleq \\
(\nu\alpha b)\,(!\,\alpha{\to}b \mid (\nu qb_1)\,(q(u).\,!\,u{\to}b_1 \mid & \\
\qquad !\,(\nu w)\,(\overline{q}\langle w\rangle.\,\llbracket\mu\beta.[\alpha]\lambda y.y\rrbracket\, w) \mid !\,b_1{\to}\alpha) \mid !\,b{\to}a) & \to (q), \sim_{\mathrm{G}}, \triangleq, \equiv \\
(\nu\alpha b)\,(!\,\alpha{\to}b \mid \llbracket\lambda y.y\rrbracket\,\alpha \mid !\,b{\to}a) \quad \triangleq, \sim_{\mathrm{R}}, \sim_{\mathrm{G}} \quad (\nu yb)\,(\llbracket y\rrbracket\, b \mid \overline{a}\langle y,b\rangle) &
\end{array}
$$

Fig. 2. The translation of a term with double output

Example 23. The translation of a β-redex reduces as:

$$
\begin{array}{ll}
\llbracket(\lambda x.P)Q\rrbracket\, a & \triangleq \\
(\nu c)\,((\nu xb)\,(\llbracket P\rrbracket\, b \mid \overline{c}\langle x,b\rangle) \mid !\,c(v,d).\,(\llbracket v{:=}Q\rrbracket \mid !d{\to}a)) & \to_{\pi} (c) \\
(\nu bx)\,(\llbracket P\rrbracket\, b \mid !\,b{\to}a \mid \llbracket x := Q\rrbracket) \mid (\nu c)\,(!\,c(v,d).\,(\llbracket v{:=}Q\rrbracket \mid !d{\to}a)) & \sim_{\mathrm{G}} \\
(\nu bx)\,(\llbracket P\rrbracket\, b \mid !\,b{\to}a \mid \llbracket x := Q\rrbracket) & \sim_{\mathrm{R}} (22) \\
(\nu x)\,(\llbracket P\rrbracket\, a \mid \llbracket x{:=}Q\rrbracket) & \triangleq \llbracket P\langle x := Q\rangle\rrbracket\, a
\end{array}
$$

This implies that β-reduction is implemented in π by at least one π-reduction.

On the other hand, μ-reduction consists of a reorganisation of the structure of a term by changing its applicative structure. Since application is essentially modelled through parallel composition, this implies that the translation of a μ-redex is essentially dealt with by congruence and renaming. For example,

$$
\begin{array}{ll}
\llbracket(\mu\beta.[\beta]P)Q\rrbracket\, a & \triangleq \\
(\nu c)\,((\nu\bullet)\,((\llbracket P\rrbracket\,\beta)[c/\beta]) \mid !\,c(v,d).\,(\llbracket v{:=}Q\rrbracket \mid !d{\to}a)) & \sim_{\mathrm{C}} \ (=_{\alpha}) \\
(\nu\beta)\,(\llbracket P\rrbracket\,\beta \mid !\,\beta(v,d).\,(\llbracket v{:=}Q\rrbracket \mid !d{\to}a)) &
\end{array}
$$

We can show, using Lem. 20, this last process is contextually equivalent to

$$
\begin{array}{l}
(\nu\gamma)\,((\nu\beta)\,(\llbracket P\rrbracket\,\gamma \mid !\,\beta(v,d).\,(\llbracket v{:=}Q\rrbracket \mid !d{\to}a)) \mid !\,\gamma(v,d).\,(\llbracket v{:=}Q\rrbracket \mid !d{\to}a)) \\
\quad \triangleq \ \llbracket P\langle\beta := Q{\cdot}a\rangle Q\rrbracket\, a
\end{array}
$$

(notice that we have separated out the outside name of the term P, being β, which we renamed to γ; this leaves two context substitutions, one dealing with the occurrences of β inside P, and one with γ[3]).

Translations of terms in $\to_{\mathbf{xH}}$-normal form are in normal form as well.

Lemma 24. $\boldsymbol{N}$ is a $\to_{\mathbf{xH}}$-nf implies $\llbracket\boldsymbol{N}\rrbracket\, a$ is irreducible.

To illustrate the expressiveness of our translation, we give some examples:

Example 25. 1. In Fig. 2 we run $\llbracket(\lambda x.x)(\mu\alpha.[\alpha](\lambda q.q)(\mu\beta.[\alpha]\lambda y.y))\rrbracket\, a$,

[3] This corresponds to the behaviour of rule ($\nwarrow$*imp-outs*) in $\mathcal{X}$.

as an example of a term that generates two outputs over α, and highlights the need for the repeated use of replication.

2. $\llbracket PQR \rrbracket a \;\triangleq,\equiv\; (\nu cc')\,(\llbracket P \rrbracket c' \mid !c'(v,d).\,(\llbracket v := Q \rrbracket \mid !d \rightarrowtriangle c) \mid$
$\qquad !c(v,d).\,(\llbracket v := R \rrbracket \mid !d \rightarrowtriangle a))$
so components of applications are placed in parallel under the translation. Similarly,
$$\llbracket M\langle\alpha := N\cdot\beta\rangle\langle\gamma := L\cdot\delta\rangle \rrbracket a \;=\; (\nu\gamma\alpha)\,(\llbracket M \rrbracket a \mid \llbracket \alpha := N\cdot\beta \rrbracket \mid \llbracket \gamma := L\cdot\delta \rrbracket)$$
so repeated structural substitutions are also placed in parallel under the translation and can be applied independently.

6 Soundness, completeness, and termination

As in [24, 31], we can now show a reduction preservation result for explicit head reduction for $\Lambda\mu\mathbf{x}$, by showing that $\llbracket\cdot\rrbracket\cdot$ preserves $\rightarrow_{\mathrm{xH}}$ up to $\rightsquigarrow^*_\pi$. Since reduction in interpreted terms takes place over hidden channels exclusively, by Lem. 8, $\rightsquigarrow^*_\pi \subseteq \sim_{\mathrm{C}}$, so we could have shown the following result using $\sim_{\mathrm{C}}$ as well, but the current formulation is more expressive; notice that we do not require the terms to be closed.

Theorem 26 (Soundness). $M \rightarrow_{\mathrm{xH}} N \Rightarrow \llbracket M \rrbracket a \rightsquigarrow^*_\pi \llbracket N \rrbracket a$.

PROOF. We show only the interesting cases.

$$\begin{array}{lll}
x\langle x := N\rangle \rightarrow N: & \llbracket x\langle x := N\rangle \rrbracket a & \triangleq \\
& (\nu x)\,(\llbracket x \rrbracket a \mid \llbracket x := N \rrbracket) & \equiv \\
& (\nu x)\,(x(u).!u \rightarrowtriangle a \mid (\nu w)\,(\overline{x}\langle w\rangle.\llbracket N \rrbracket w) \mid \llbracket x := N \rrbracket) & \rightarrow_\pi \quad (x) \\
& (\nu w)\,(!w \rightarrowtriangle a \mid \llbracket N \rrbracket w) \mid (\nu x)\,(\llbracket x := N \rrbracket) & \sim_{\mathrm{R}}, \sim_{\mathrm{G}} \quad \llbracket N \rrbracket a
\end{array}$$

$$\begin{array}{lll}
(PQ)\langle x := N\rangle \rightarrow (P\langle x := N\rangle Q)\langle x := N\rangle,\ x = hv(P): & \llbracket (PQ)\langle x := N\rangle \rrbracket a & \triangleq \\
& (\nu x)\,((\nu c)\,(\llbracket P \rrbracket c \mid !\llbracket c := Q\cdot a \rrbracket) \mid \llbracket x := N \rrbracket) & \sim_{\mathrm{C}}\ (20) \\
& (\nu x)\,((\nu c)\,((\nu x)\,(\llbracket P \rrbracket c \mid \llbracket x := N \rrbracket) \mid !\llbracket c := Q\cdot a \rrbracket) \mid \llbracket x := N \rrbracket) & \triangleq \\
& (\nu x)\,((\nu c)\,(\llbracket P\langle x := N\rangle \rrbracket c \mid !\llbracket c := Q\cdot a \rrbracket) \mid \llbracket x := N \rrbracket) & \triangleq,\equiv \\
& (\nu x)\,(\llbracket P\langle x := N\rangle Q \rrbracket a \mid \llbracket x := N \rrbracket) & \triangleq \\
& \llbracket (P\langle x := N\rangle Q)\langle x := N\rangle \rrbracket a &
\end{array}$$

$$\begin{array}{lll}
(\mu\beta.M)N \rightarrow \mu\gamma.M\langle\beta := N\cdot\gamma\rangle,\ \gamma\ \mathit{fresh}: & \llbracket (\mu\beta.M)N \rrbracket a & \triangleq \\
& (\nu c)\,((\nu\bullet)\,((\llbracket M \rrbracket \bullet)[c/\beta]) \mid !c(v,d).\,(\llbracket v := N \rrbracket \mid !d \rightarrowtriangle a)) & =_\alpha \\
& (\nu\beta)\,((\nu\bullet)\,(\llbracket M \rrbracket \bullet) \mid !\beta(v,d).\,(\llbracket v := N \rrbracket \mid !d \rightarrowtriangle a)) & \equiv,= \\
& (\nu\bullet)\,(((\nu\beta)\,(\llbracket M \rrbracket \bullet \mid !\beta(v,d).\,(\llbracket v := N \rrbracket \mid !d \rightarrowtriangle \gamma)))[a/\gamma]) & \triangleq \\
& \llbracket \mu\gamma.M\langle\beta := N\cdot\gamma\rangle \rrbracket a &
\end{array}$$

$$\begin{array}{lll}
([\alpha]M)\langle\alpha := N\cdot\gamma\rangle \rightarrow [\gamma](M\langle\alpha := N\cdot\gamma\rangle)N: & \llbracket ([\alpha]M)\langle\alpha := N\cdot\gamma\rangle \rrbracket & \triangleq \\
& (\nu\alpha)\,(\llbracket M \rrbracket \alpha \mid !\alpha(v,d).\,(\llbracket v := N \rrbracket \mid !d \rightarrowtriangle \gamma)) & \sim_{\mathrm{C}}\ (20) \\
& (\nu c)\,((\nu\alpha)\,(\llbracket M \rrbracket c \mid !\alpha(v,d).\,(\llbracket v := N \rrbracket \mid !d \rightarrowtriangle \gamma)) \mid & \\
& \qquad !c(v,d).\,(\llbracket v := N \rrbracket \mid !d \rightarrowtriangle \gamma)) & \triangleq \\
& (\nu c)\,(\llbracket M\langle\alpha := N\cdot\gamma\rangle \rrbracket c \mid !c(v,d).\,(\llbracket v := N \rrbracket \mid !d \rightarrowtriangle \gamma)) & \triangleq \\
& \llbracket [\gamma]M\langle\alpha := N\cdot\gamma\rangle N \rrbracket &
\end{array}$$

□

The main soundness result is formulated as:

Theorem 27 (Operational Soundness for $\to_{\mathbf{x}\mathrm{H}}$). 1. $M \to^*_{\mathbf{x}\mathrm{H}} N \Rightarrow \llbracket M \rrbracket a \rightsquigarrow^*_\pi \llbracket N \rrbracket a$.
2. $M \uparrow_{\mathbf{x}\mathrm{H}} \Rightarrow \llbracket M \rrbracket a \uparrow_\pi$.

Since $\rightsquigarrow^*_\pi \subseteq \sim_{\mathrm{C}}$, which is symmetric, Thm. 27 gives that $\llbracket \cdot \rrbracket \cdot$ preserves $=_{\mathbf{x}\mathrm{H}}$ up to $\sim_{\mathrm{C}}$.

Corollary 28 (Adequacy). $M =_{\mathbf{x}\mathrm{H}} N \Rightarrow \llbracket M \rrbracket a \sim_{\mathrm{C}} \llbracket N \rrbracket a$.

This result states that our encoding gives, in fact, a semantics for the explicit head reduction for $\Lambda\mu$. As for a full abstraction result, note that we cannot show the reverse of Cor. 28, since different unsolvable terms like $(\lambda x.xx)(\lambda x.xx)$ and $(\lambda w.www)(\lambda w.www)$ are not equivalent under $=_{\mathbf{x}\mathrm{H}}$, but are *contextually* equivalent under $\llbracket \cdot \rrbracket \cdot$, *i.e.* have the same observable behaviour, as is illustrated by the fact that their translations never exhibit an output.

We can also show operational completeness for $\to_{\mathbf{x}\mathrm{H}}$.

Theorem 29 (Operational completeness for $\to_{\mathbf{x}\mathrm{H}}$). If $\llbracket M \rrbracket a \to_\pi P$ then there exists N such that $P \rightsquigarrow^*_\pi \llbracket N \rrbracket a$, and $M \to_{\mathbf{x}\mathrm{H}} N$.

This in turn can be used to show:

Lemma 30. 1. Let M be a term in $\Lambda\mu\mathbf{x}$. If $\llbracket M \rrbracket a \to^*_\pi \llbracket N \rrbracket a$ then $M \to^*_{\mathbf{x}\mathrm{H}} N$.
2. Let $M \in \Lambda\mu$, *i.e.* a (pure) $\Lambda\mu$-term. If $\llbracket M \rrbracket a \to_\pi P$ then there exists $N \in \Lambda\mu\mathbf{x}$ and $L \in \Lambda\mu$ such that $P \sim_{\mathrm{C}} \llbracket N \rrbracket a$, and $M \to^*_{\mathbf{x}\mathrm{H}} N$ and $N \to^*_{:=} L$.

We can show the following termination results:

Theorem 31 (Termination). 1. If $M \to^*_{\mathbf{x}\mathrm{H}} N$, with N in explicit head-normal from, then $\llbracket M \rrbracket a \downarrow_\pi$.
2. If $M \to^*_{\beta\mu} N$, with N in head-normal from, then $\llbracket M \rrbracket a \downarrow_\pi$.
3. Let $M \in \Lambda\mu$. If $\llbracket M \rrbracket a \downarrow_\pi$ then there exists $N \in \Lambda\mu\mathbf{x}$ and L in $\to_{\lambda\mu}$-head normal form such that $\llbracket M \rrbracket a \sim_{\mathrm{C}} \llbracket N \rrbracket a$, and $M \to^*_{\mathbf{x}\mathrm{H}} N$ and $N \to^*_{:=} L$.

Notice that, in the first case, the normal form of $\llbracket M \rrbracket a$ need not be $\llbracket N \rrbracket a$; a similar observation can be made with respect to Milner's encoding. Notice also that this result is stronger than the formulation of the termination result for Milner's encoding in [31], since it models reduction to head-normal form, not just normal form. However, since terms that have a normal form have a head-normal form as well, Thm. 31 immediately leads to:

Corollary 32. If $M \downarrow_{\beta\mu}$, then $\llbracket M \rrbracket a \downarrow_\pi$.

Conclusions

We have defined an output based, logic inspired translation of untyped $\Lambda\mu$ with explicit substitution into the π-calculus and shown that it respects step-by-step head-reduction, assignable types, head-conversion, and termination. We conjecture that we can show the results shown above also for head reduction with *implicit* substitution; for this we

would need to show that, if $M \to^*_{:=} N$, then $\llbracket M \rrbracket a \sim_{\mathrm{C}} \llbracket N \rrbracket a$. It seems that the approach via Levy-Longo trees is more suitable for that.

There are many alternatives to the approach we have chosen to follow here; especially our choice for contextual equivalence (inspired by λ-calculus semantics) could be replaced by branching semantics, or a bisimulation-like equivalence. The natural question is then, which of our properties would be affected? Would branching and non-branching equivalences to coincide, maybe by exploiting some confluence properties?

We leave these issues for future work.

References

1. M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit Substitutions. *JFP*, 1(4):375–416, 1991.
2. M. Abadi and A. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *CCS'97*, pp 36–47, 1997.
3. S. Abramsky. The lazy lambda calculus. *Research topics in functional programming*, pages 65–116. Addison-Wesley, 1990.
4. P. Audebaud. Explicit Substitutions for the $\Lambda\mu$ Calculus. RR 94-26, ÉNS de Lyon, 1994.
5. S. van Bakel. Completeness and Partial Soundness Results for Intersection & Union Typing for $\overline{\lambda}\mu\tilde{\mu}$. *Annals of Pure and Applied Logic*, 161:1400–1430, 2010.
6. S. van Bakel. Completeness and Soundness results for $\mathcal{X}$ with Intersection and Union Types. *Fundamenta Informaticae*, 2012. To appear.
7. S. van Bakel, F. Barbanera, and U' de'Liguoro. A Filter Model for $\lambda\mu$. TLCA'11, LNCS 6690, pp. 213–228, 2011.
8. S. van Bakel, L. Cardelli, and M.G. Vigliotti. From $\mathcal{X}$ to π; Representing the Classical Sequent Calculus in the π-calculus. *CL&C'08*, 2008.
9. S. van Bakel and P. Lescanne. Computation with Classical Sequents. *MSCS*, 18:555–609, 2008.
10. S. van Bakel and M.G. Vigliotti. A logical interpretation of the λ-calculus into the π-calculus, preserving spine reduction and types. *CONCUR'09*, LNCS 5710, pp 84 – 98, 2009.
11. H. Barendregt. *The Lambda Calculus: its Syntax and Semantics*. North-Holland, 1984.
12. R. Bloo and K.H. Rose. Preservation of Strong Normalisation in Named Lambda Calculi with Explicit Substitution and Garbage Collection. *CSN'95*, pp 62–72, 1995.
13. N.G. de Bruijn. Lambda Calculus Notation with Nameless Dummies: A Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem. *Ind. Math.*, 34:381–392, 1972.
14. A. Church. A Note on the Entscheidungsproblem. *JSL*, 1(1):40–41, 1936.
15. M. Cimini, C. Sacerdoti Coen, and D. Sangiorgi. $\overline{\lambda}\mu\tilde{\mu}$ calculus, π-calculus, and abstract machines. *EXPRESS'09*, 2009.
16. T. Crolard. A confluent lambda-calculus with a catch/throw mechanism. *JFP*, 9(6):625–647, 1999.
17. P.-L. Curien and H. Herbelin. The Duality of Computation. *ICFP'00*, pp 233–243. ACM, 2000.
18. G. Gentzen. Investigations into logical deduction. *The Collected Papers of Gerhard Gentzen*. Ed M. E. Szabo, North Holland, 68ff (1969), 1935.
19. Ph. de Groote. On the relation between the $\lambda\mu$-calculus and the syntactic theory of sequential control. *LPAR'94*, LNCS 822, pp 31–43, 1994.
20. H. Herbelin. *C'est maintenant qu'on calcule: au cœur de la dualité*. Mémoire d'habilitation, Université Paris 11, 2005.
21. K. Honda and M. Tokoro. An object calculus for asynchronous communication. *ECOOP'91*, LNCS 512, pp 133–147, 1991.
22. K. Honda, N. Yoshida, and M. Berger. Control in the π-Calculus. *CW'04*, 2004.
23. S.B. Lassen. Head Normal Form Bisimulation for Pairs and the $\lambda\mu$-Calculus. *LICS'06*, pp 297–306, 2006.
24. R. Milner. Functions as processes. *MSCS*, 2(2):269–310, 1992.
25. M. Parigot. An algorithmic interpretation of classical natural deduction. *LPAR'92*, LNCS 624, pp 190–201, 1992.
26. M. Parigot. Strong Normalization for Second Order Classical Natural Deduction. *LICS'93*, pp 39–46, 1993.
27. B.C. Pierce and D. Sangiorgi. Typing and Subtyping for Mobile Processes. *MSCS*, 6(5):409–453, 1996.
28. W. Py. *Confluence en $\lambda\mu$-calcul*. Phd thesis, Univ. Savoie, 1998.
29. D. Sangiorgi. *Expressing Mobility in Process Algebra: First Order and Higher Order Paradigms*. PhD thesis, Univ. Edinburgh, 1992.
30. D. Sangiorgi. Lazy functions and mobile processes. RR 2515, INRIA, Sophia-Antipolis, France, 1995.
31. D. Sangiorgi and D. Walker. *The Pi-Calculus*. Cambridge University Press, 2001.
32. P. Sestoft. Demonstrating Lambda Calculus Reduction. *The Essence of Computation*, LNCS 2566, pp 420–435, 2001.
33. C.P. Wadsworth. The relation between computational and denotational properties for Scott's D_∞-models of the lambda-calculus. *SIAM JoC*, 5:488–521, 1976.