An Investigation of Linear Substitution λ-Calculus as Session-Typed Processes

S. Alves
DCC-FCUP & CRACS
University of Porto
Porto, Portugal

D. Nantes-Sobrinho
Departamento de Matemática
Universidade de Brasília
Brasília, Brazil

J.A. Pérez
Bernoulli Institute
University of Groningen
The Netherlands

D. Ventura
Instituto de Informática
Universidade Federal de Goiás
Goiânia, Brazil

Abstract

This work in progress investigates a type directed encoding of the *simply-typed* linear substitution λ-calculus to session-typed π-processes, taking into account an operational semantics defined by a linear weak head reduction strategy. The encoding adapts the one by Toninho et. al., which encodes the simply-typed λ-calculus into a session-typed π-calculus through linear logic, while extending it to explicit substitutions, considering a generalisation of the π-calculus operational semantics with reduction rules at *a distance*, as proposed by Accattoli. These changes, while weakening the simulation of λ-calculus in the π-calculus, when compared with the 1-to-1 simulation obtained by Accattoli, allow for a tight encoding of well-typed functions as session-typed processes.

Keywords: Lambda Calculus, Pi-Calculus, Encoding, Session Types, Simulation

1 Introduction

This paper proposes an in-depth investigation on encodings from *functions to processes*, focusing on works proposed by Accattoli [1] and by Toninho et. al. [13].

Ever since Milner's first encoding of the λ-calculus [9], motivated by the desire to establish a canonical model for the concurrent paradigm in the same way that the λ-calculus provides a canonical model for the functional paradigm, there have been a variety of subsequent encodings for the various extensions of the π-calculus, like the ones in [10,11,12,5,4], just to mention a few.

Inspired by Milner's encodings of the λ-calculus, Accattoli [1] proposed a tight correspondence between a λ-calculus with linear substitutions and the π-calculus, giving a strong simulation with respect to reduction. In order to obtain such an encoding, explicit substitutions *at a distance* were used, however, only the untyped version of λ_{lsub} was considered. In the realm of simply typed terms, Toninho et. al [13], explored the relation between simply typed terms and session typed processes, through yet another encoding on functions as processes based on a linear encoding of terms in linear logic.

In this paper, we combine both approaches to obtain a type-directed encoding of a linear substitution λ-calculus, named λ_{lsub}-calculus, into session typed π-processes, in which the reduction strategy is the *linear weak head reduction* that corresponds closely to evaluation in the π-calculus. The idea is to explore the qualities of both works, to have an encoding in which the simulation of λ-calculus in the π-calculus is as tight as it can be, as in [1]; on the other hand, we also wish to maintain the correspondence between well-typed functions as session-typed processes, as in [13]. There is however a mismatch between their encodings: while the former considers the linear substitution λ_{lsub} calculus with explicit substitutions at a distance and an asynchronous fragment of the π-calculus with unary and binary input and outputs, the latter works with simply typed linear λ-terms, and an synchronous fragment of the π-calculus, which contains only unary input/output in addition to the *forward construct* $[x \leftrightarrow y]$ that equates channel names x and y.

Two approaches were considered: to use Acattoli's encoding and operational semantics, and develop a new strategy to connect the simply typed λ_{lsub} to session typed processes; an alternative is to extend the encoding by Toninho et. al., which already has the connection with session types, by adding explicit substitutions at a

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

distance and investigate what is obtained from the weak head linear strategy of reduction. In this work we have opted for the latter approach. Although we could inherit a lot from both works, the small changes made on the encoding and on the operational semantics caused interesting modifications in the final result. To begin, we consider a simple types discipline for λ_{lsub}. To follow the weak head linear reduction strategy we extended the notions of evaluation/substitution contexts to linear λ-calculus. A translation $[\![\cdot]\!]_a$ from typed λ_{lsub}-calculus to the typed linear λ-calculus is extended with contexts and explicit substitutions. Technical results regarding operational correspondence and dynamic soundness are presented, it is worthy mentioning that their proofs follow the techniques in [1], due to the semantics involved. The encoding proposed in [13] now extends to explicit substitutions; therefore, all the intermediate translations that the authors used had to be extended to deal with explicit substitutions. The simulation results are not as tight as initially expected (Theorems 4.4 and 4.7). This is because of the forward construct, which induces some extra computation steps, resulting in a scenario where one λ-reduction step corresponds to three π-reduction steps.

Organisation.
Section 2 introduces the syntax and semantics of linear substitution calculus λ_{lsub}. Some properties of the calculus are given. Section 3 gives the fragment of the π-calculus that we will work with, operational semantics and some syntactic properties. In Section 4 we give the encoding from λ_{lsub} to π-calculus and then some simulation results. Finally, Section 5 concludes the paper.

2 Preliminaries

In this section we present a simply typed version of the *linear substitution calculus* λ_{lsub}. Following the steps of [13], a *linear version* was used as an intermediate language in obtaining the encoding to session typed processes. For the sake of clarity, the corresponding $\hat{\lambda}_{lsub}$-calculus is included in the full report [2]. Basic notions of the λ-calculus are assumed, for more details we refer the reader to [3].

2.1 Simply typed linear substitution calculus λ_{lsub}

λ_{lsub} is given by the following grammar of terms M, N and types τ, σ, ρ, where α denotes *base types*. Typing judgments have the form $\Gamma \vdash M : \tau$, where the typing context Γ consists of a finite set of (term) variable type assignments of the form $x : \sigma$, where any variable occurs at most once as subject. *Typing derivation rules* are given in Figure 1.

$$\begin{array}{ll} \text{(Terms)} & M, N ::= \quad x \mid \lambda x.M \mid MN \mid M[N/x] \\ \text{(Types)} & \tau, \sigma, \rho ::= \alpha \mid \sigma \to \tau \end{array}$$

The constructor $M[N/x]$ is called *explicit substitution* (of N for x in M), the implicit substitution is denoted by $M\{N/x\}$. The set $\mathtt{fv}(M)$ of free variables of M is defined as usual where, in particular, $\mathtt{fv}(M[N/x]) = (\mathtt{fv}(M) \setminus \{x\}) \cup \mathtt{fv}(N)$.

$$\frac{}{\Gamma, x{:}\tau \vdash x : \tau}\ (\mathtt{ax}) \qquad \frac{\Gamma \vdash N : \sigma \qquad \Gamma, x{:}\sigma \vdash M : \tau}{\Gamma \vdash M[N/x] : \tau}\ (\mathtt{cut})$$

$$\frac{\Gamma, x{:}\sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}\ (\to \mathtt{i}) \qquad \frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,N : \tau}\ (\to \mathtt{e})$$

Fig. 1. Typing Rules for λ_{lsub}

Contexts and Operational Semantics for λ_{lsub}.
An *evaluation context*, denoted by $\mathbb{E}_S$, is defined according to the following grammar:

$$\mathbb{E}_\emptyset := (\![\,\cdot\,]\!) \mid \mathbb{E}_\emptyset M \qquad \mathbb{E}_{S \uplus \{x\}} := \mathbb{E}_S[M/x] \mid \mathbb{E}_{S \uplus \{x\}} M$$

A *substitution context*, denoted by $\mathbb{L}_S$, is given by:

$$\mathbb{L}_\emptyset := (\![\,\cdot\,]\!) \qquad \mathbb{L}_{S \uplus \{x\}} := \mathbb{L}_S[M/x]$$

In both cases, S denotes the set of variables captured by the context. Evaluation contexts will range over $\mathbb{E}, \mathbb{E}', \mathbb{E}'', \ldots$ whereas substitution contexts will range over $\mathbb{L}, \mathbb{L}', \mathbb{L}'', \ldots$. For instance, the list context $(\![\,\cdot\,]\!)[M_1/x][M_2/y]$ can be denoted by $\mathbb{L}_{\{x,y\}}(\![\,\cdot\,]\!)$, where $\mathbb{L}_{\{x,y\}}(\!|N|\!)$ denotes $N[M_1/x][M_2/y]$.

Operational semantics is defined by the *linear weak head reduction* $\multimap$ given as the union of closure by evaluation contexts of the rules $\multimap_{dB}$ and $\multimap_{ls}$:

$$\mathbb{L}_S([\lambda x.M])N \multimap_{dB} \mathbb{L}_S([M[N/x]]) \qquad \mathbb{E}_S([x])[N/x] \multimap_{ls} \mathbb{E}_S([N])[N/x], \text{ if } x \notin S$$

Both rules assume that $\mathtt{fv}(N) \cap S = \emptyset$.

In order to establish soundness of type assignments, proving the subject reduction property, a linear) substitution lemma is proved beforehand as usual, being a particular case of partial substitutions[8].

Lemma 2.1 (Subject Reduction) *If* $\Gamma \vdash M : \tau$ *and* $M \multimap M'$ *then* $\Gamma \vdash M' : \tau$.

Proof. By induction on $\multimap$, with a Linear Substitution Lemma. □

3 The π-calculus

We consider a fragment of synchronous π-calculus without sums, with the forwarding process:

$$P, Q \quad ::= x\langle y\rangle.P \mid (\nu x)P \mid x(y).P \mid !x(y).P \mid (P \mid Q) \mid [x \leftrightarrow y]$$

The set $\mathtt{fn}(P)$ of free name of a process P is defined as expected, considering (νx) and $x(y)$ as binders. In other words, $\mathtt{fn}((\nu x)P) = \mathtt{fn}(P) \setminus \{x\}$ and $\mathtt{fn}(x(y).P) = (\mathtt{fn}(P) \setminus \{y\}) \cup \{x\}$.

A *non-blocking context*, range over $\mathbb{N}, \mathbb{N}', \mathbb{N}'', \ldots$, is also parameterized by a set S of variables, and defined by:

$$\mathbb{N}_\emptyset := ([\,\cdot\,]) \mid (\mathbb{N}_\emptyset \mid Q) \mid (P \mid \mathbb{N}_\emptyset) \qquad \mathbb{N}_{S \uplus \{x\}} := (\nu x)\mathbb{N}_S \mid \mathbb{N}_\emptyset([\mathbb{N}_{S \uplus \{x\}}])$$

Structural congruence $\equiv$ is the least congruence generated by the following rules and closed by non-blocking contexts.

$$\begin{array}{lll} P \mid (Q \mid R) \equiv (P \mid Q) \mid R & (\nu x)(\nu y)P \equiv (\nu y)(\nu x)P & P \mid Q \equiv Q \mid P \\ x \notin \mathtt{fn}(P) \Rightarrow P \mid (\nu x)Q \equiv (\nu x)(P \mid Q) & P \equiv_\alpha Q \;\Rightarrow\; P \equiv Q & [x \leftrightarrow y] \equiv [y \leftrightarrow x] \end{array}$$

Operational Semantics

The *reduction relation* $\to_\pi$ is defined by the rewriting rules, closed by non-blocking contexts modulo $\equiv$:

$$\begin{array}{rl} x\langle y\rangle.Q \mid x(z).P & \to_\tau \; Q \mid P\{y/z\} \\ x\langle y\rangle.Q \mid !x(z).P & \to_{!\tau} \; Q \mid P\{y/z\} \mid !x(z).P \\ (\nu x)(P \mid [x \leftrightarrow y]) & \to_r \; P\{y/x\} \quad (\text{provided } x \neq y) \end{array}$$

Pi-calculus at a distance.

Consider the reduction rule $\Rightarrow_\pi$ given by the closure by non-blocking contexts of the relation: if $x \notin S \cup V \cup W$, then

$$\begin{array}{rl} \mathbb{N}_S([x\langle y\rangle.Q]) \mid \mathbb{N}'_V([x(z).P]) & \Rightarrow_\tau \; \mathbb{N}'_V([\mathbb{N}_S([Q \mid P\{y/z\}])]) \\ \mathbb{N}_S([x\langle y\rangle.Q]) \mid \mathbb{N}'_V([!x(z).P]) & \Rightarrow_{!\tau} \; \mathbb{N}'_V([\mathbb{N}_S([Q \mid P\{y/z\} \mid !x(z).P])]) \\ \mathbb{N}_{S \uplus \{x\}}([\mathbb{N}'_V([P]) \mid \mathbb{N}''_W([[x \leftrightarrow y]])]) & \Rightarrow_r \; \mathbb{N}_S([\mathbb{N}'_V([\mathbb{N}''_W([P])])\{y/x\}]) \end{array}$$

The reduction rules in $\Rightarrow_\pi$ take in account the forward construct. In addition, there are conditions on the variables which are implicit from the Barendregt's Convention on bound variables: $S \cap V = \emptyset$, $S \cap \mathtt{fn}(P) = \emptyset$ and $\mathtt{fn}(\mathbb{N}_S) \cap V = \emptyset$.

Lemma 3.1 *Let* S *be a set of variables,* $\mathbb{N}_S$ *a non-blocking context,* P *a process such that* $\mathtt{fn}(P) \cap S = \emptyset$, *and* $x, y \notin S$. *Then*

(i) $\mathbb{N}_S([Q]) \mid P \equiv \mathbb{N}_S([Q \mid P])$.

(ii) *If* $x \notin \mathtt{fn}(\mathbb{N}_S)$ *then* $(\nu x)\mathbb{N}_S([P]) \equiv \mathbb{N}_S([(\nu x)P])$.

Lemma 3.2 (Harmony Lemma) *Let* P *and* Q *be processes:*

$$\frac{}{\Gamma; x{:}\tau \Rightarrow [x \leftrightarrow a] :: a : \tau}\ (\mathtt{id}) \qquad \frac{\Gamma; \Delta, x{:}\sigma \Rightarrow P :: a : \tau}{\Gamma; \Delta \Rightarrow a(x).P :: a : \sigma \multimap \tau}\ (\multimap \mathtt{R})$$

$$\frac{\Gamma; \emptyset \Rightarrow P :: b : \tau}{\Gamma; \emptyset \Rightarrow\ !a(b).P :: a :!\tau}\ (!\mathtt{R}) \qquad \frac{\Gamma, u : \sigma; \Delta \Rightarrow P :: a : \rho}{\Gamma; \Delta, b :!\sigma \Rightarrow P\{b/u\} :: a : \rho}\ (!\mathtt{L})$$

$$\frac{\Gamma; \Delta \Rightarrow P :: b : \sigma \qquad \Gamma; \Delta', x : \tau \Rightarrow Q :: a : \rho}{\Gamma; \Delta, \Delta', x : \sigma \multimap \tau \Rightarrow (\nu b)x\langle b\rangle.(P \mid Q) :: a : \rho}\ (\multimap \mathtt{L}) \qquad \frac{\Gamma, u : \sigma; \Delta, x : \sigma \Rightarrow P :: a : \rho}{\Gamma, u : \sigma; \Delta \Rightarrow (\nu x)u\langle x\rangle.P :: a : \rho}\ (\mathtt{copy})$$

$$\frac{\Gamma; \Delta \Rightarrow P :: b : \sigma \qquad \Gamma; \Delta', x{:}\sigma \Rightarrow Q :: a : \tau}{\Gamma; \Delta, \Delta' \Rightarrow (\nu b)(P \mid Q) :: a : \tau}\ (\mathtt{cut}) \qquad \frac{\Gamma; \emptyset \Rightarrow P :: b : \sigma \qquad \Gamma, u : \sigma; \Delta \Rightarrow Q :: a : \tau}{\Gamma; \Delta \Rightarrow (\nu u)(!u\langle b\rangle.P \mid Q) :: a : \tau}\ (\mathtt{cut}^!)$$

Fig. 2. Session Typing Rules

(i) *There exists P' such that $P \equiv P' \Rightarrow_\pi Q$ iff there exists Q' such that $P \Rightarrow_\pi Q' \equiv Q$.*
(ii) *$P \rightarrow_\pi Q$ iff there exists Q' such that $P \Rightarrow_\pi Q' \equiv Q$.*

4 Functions as Session Typed Processes

In the present section we relate simply typed terms in the λ_{lsub}-calculus to its corresponding encoding as a session typed process. The encoding is derived following [13], in which intermediate steps are omitted. It is worth noticing that, in an encoding of λ_{lsub} in a linear calculus, named $\hat{\lambda}_{lsub}$, explicit substitution translations were understood through terms already present in the linear calculus of [13] and evaluation/list contexts encodings were straightforward. Having said that, the translation from $\hat{\lambda}_{lsub}$ to session typed processes has no essential difference from [13] (see [2] for further details).

Remark 1 *As for the original encoding of the λ-calculus into the π-calculus given by Milner, the encoding of λ_{lsub} into π-calculus is parameterized by a channel name a as in [1] we will assume that these special names are taken from a set $\mathcal{A}$ which is disjoint from the set of variable names V, and whose elements are denoted by $a, b, c, \ldots$.*

A typing judgement for processes is of the form $\Gamma; \Delta \Rightarrow P :: a : A$, where P is a process that implements a session of type A on channel a, based on assignments of types to channel names given by the contexts Γ and Δ. Channel names in Γ, Δ and a must be mutually distinct, renaming of bound names are assumed when necessary. The standard session typing rules are given in Figure 2 are taken from [6].

4.1 Encoding of λ_{lsub} into π-calculus

The encoding $[\![\cdot]\!]_a$ from λ_{lsub} to π-calculus below, is basically the same encoding $[\![\cdot]\!]_z$ given in [13], extended with explicit substitutions [1].

$$\begin{aligned}
[\![x]\!]_a &= (\nu y)u_x\langle y\rangle.[y \leftrightarrow a] \\
[\![\lambda x.M]\!]_a &= a(x).(\nu y)([x \leftrightarrow y] \mid [\![M]\!]_a\{y/u_x\}) \\
[\![MN]\!]_a &= (\nu b)([\![M]\!]_b \mid (\nu c)b\langle c\rangle.((!c(y).[\![N]\!]_y) \mid [b \leftrightarrow a])) \\
[\![M[N/x]]\!]_a &= (\nu b)([\![M]\!]_a\{b/u_x\} \mid !b(c).[\![N]\!]_c)
\end{aligned}$$

The example below illustrates the encoding of the λ-term $(\lambda x.M)N$ as well as some reduction steps:

[1] variables named u_x correspond to unrestricted variables in the $\hat{\lambda}_{lsub}$ linear calculus, encoding a term variable x from λ_{lsub}

Example 4.1 For reduction:

$$\begin{array}{rcll}
[\![(\lambda x.M)N]\!]_a & = & (\nu b)([\![\lambda x.M]\!]_b \mid (\nu y)b\langle y\rangle.((!y(c).[\![N]\!]_c) \mid [b \leftrightarrow a])) & (1)\\
& = & (\nu b)(b(x).(\nu y')([x \leftrightarrow y'] \mid [\![M]\!]_b\{y'/u_x\}) \mid (\nu y)b\langle y\rangle.((!y(c).[\![N]\!]_c) \mid [b \leftrightarrow a])) & (2)\\
& \equiv & (\nu b)(\nu y)(b(x).(\nu y')([x \leftrightarrow y'] \mid [\![M]\!]_b\{y'/u_x\}) \mid b\langle y\rangle.((!y(c).[\![N]\!]_c) \mid [b \leftrightarrow a])) & (3)\\
& \equiv & (\nu b)(\nu y)(b\langle y\rangle.((!y(c).[\![N]\!]_c) \mid [b \leftrightarrow a]) \mid b(x).(\nu y')([x \leftrightarrow y'] \mid [\![M]\!]_b\{y'/u_x\})) & (4)\\
& \Rightarrow_\tau & (\nu b)(\nu y)((!y(c).[\![N]\!]_c) \mid [b \leftrightarrow a] \mid (\nu y')([y \leftrightarrow y'] \mid [\![M]\!]_b\{y'/u_x\})) & (5)\\
& \equiv & (\nu y)((!y(c).[\![N]\!]_c) \mid (\nu b)((\nu y')([y \leftrightarrow y'] \mid [\![M]\!]_b\{y'/u_x\}) \mid [b \leftrightarrow a])) & (6)\\
& \equiv & (\nu y)(\nu b)(\nu y')((!y(c).[\![N]\!]_c) \mid [y \leftrightarrow y'] \mid [\![M]\!]_b\{y'/u_x\} \mid [b \leftrightarrow a]) & (7)\\
& \equiv & (\nu y')((\nu y)((!y(c).[\![N]\!]_c) \mid [y \leftrightarrow y']) \mid (\nu b)([\![M]\!]_b\{y'/u_x\} \mid [b \leftrightarrow a])) & (8)\\
& \Rightarrow_r & (\nu y')((!y'(c).[\![N]\!]_c) \mid (\nu b)([\![M]\!]_b\{y'/u_x\} \mid [b \leftrightarrow a])) & (9)\\
& \Rightarrow_r & (\nu y')((!y'(c).[\![N]\!]_c) \mid [\![M]\!]_a\{y'/u_x\}) & (10)
\end{array}$$

As expected, the encoding preserves typability.

Theorem 4.2 *If* $\Gamma \vdash M : \tau$ *then* $\langle\!\langle\Gamma\rangle\!\rangle; \emptyset \Rightarrow [\![M]\!]_a :: a : \langle\!\langle\tau\rangle\!\rangle$

Proof. Induction on $\Gamma \vdash M : \tau$. We present the case for (`cut`) rule: $\dfrac{\Gamma \vdash N : \sigma \qquad x{:}\sigma, \Gamma \vdash M : \tau}{\Gamma \vdash M[N/x] : \tau}$

By *i.h.* we have both $\langle\!\langle\Gamma\rangle\!\rangle; \emptyset \Rightarrow [\![N]\!]_b :: b : \langle\!\langle\sigma\rangle\!\rangle$ and $u_x{:}\langle\!\langle\sigma\rangle\!\rangle, \langle\!\langle\Gamma\rangle\!\rangle; \emptyset \Rightarrow [\![M]\!]_a :: a : \langle\!\langle\tau\rangle\!\rangle$. Therefore,

$$\dfrac{\dfrac{\langle\!\langle\Gamma\rangle\!\rangle; \emptyset \Rightarrow [\![N]\!]_b :: b : \langle\!\langle\sigma\rangle\!\rangle}{\langle\!\langle\Gamma\rangle\!\rangle; \emptyset \Rightarrow !y(b).[\![N]\!]_b :: b : !\langle\!\langle\sigma\rangle\!\rangle} \qquad \dfrac{u_x{:}\langle\!\langle\sigma\rangle\!\rangle, \langle\!\langle\Gamma\rangle\!\rangle; \emptyset \Rightarrow [\![M]\!]_a :: a : \langle\!\langle\tau\rangle\!\rangle}{\langle\!\langle\Gamma\rangle\!\rangle; y : !\langle\!\langle\sigma\rangle\!\rangle \Rightarrow [\![M]\!]_a\{y/u_x\} :: a : \langle\!\langle\tau\rangle\!\rangle}}{\langle\!\langle\Gamma\rangle\!\rangle; \emptyset \Rightarrow (\nu y)(!y(b).[\![N]\!]_b \mid [\![M]\!]_a\{y/u_x\}) :: a : \langle\!\langle\tau\rangle\!\rangle}$$

□

Regarding subject reduction, not all terms in structural-equivalent classes of terms are typable. We illustrate this issue in the example below.

Example 4.3 [Cont. Example 4.1]
Let $\Gamma \vdash (\lambda x.M)N : \tau$. Then $\langle\!\langle\Gamma\rangle\!\rangle; \emptyset \Rightarrow (\nu b)([\![\lambda x.M]\!]_b \mid (\nu y)b\langle y\rangle.((!y(c).[\![N]\!]_c) \mid [b \leftrightarrow a])) :: a : \langle\!\langle\tau\rangle\!\rangle$, from Thm. 4.2, where $[\![(\lambda x.M)N]\!]_a = (\nu b)([\![\lambda x.M]\!]_b \mid (\nu y)b\langle y\rangle.((!y(c).[\![N]\!]_c) \mid [b \leftrightarrow a]))$ is the term in (1)/(2).
When reducing from term in (4) to term in (5), the latter is not typable in the session system. On the other hand, term in (6) is typable in the system:

$$\langle\!\langle\Gamma\rangle\!\rangle; \emptyset \Rightarrow (\nu y)((!y(c).[\![N]\!]_c) \mid (\nu b)((\nu y')([y \leftrightarrow y'] \mid [\![M]\!]_b\{y'/u_x\}) \mid [b \leftrightarrow a])) :: a : \langle\!\langle\tau\rangle\!\rangle$$

.

4.2 Simulation Results

In this subsection we show that simulation of one step in the λ_{lsub}-calculus demands exactly 3 steps of the π-calculus in the corresponding encoding (Theorems 4.4 and 4.7), while completeness is obtained directly instead of relying in some sort of *substitution lifting* as introduced in [13] [2]. In order to obtain such results, technicalities regarding the interaction between contexts and reduction had to be proven. Complete proofs are available in [2].

Theorem 4.4 ($\rightarrow_\pi$ 3-simulates $\multimap$ via $[\![]\!]_a$)

(i) $M' \multimap_{\mathtt{dB}} N$ *implies* $[\![M']\!]_a \Rightarrow_\tau \overset{2}{\Rightarrow}_r \equiv [\![N]\!]_a$.
(ii) $M' \multimap_{\mathtt{ls}} N$ *implies* $[\![M']\!]_a \Rightarrow_{!\tau} \Rightarrow_r \equiv [\![N]\!]_a$.

Proof.

[2] Definition 3.2 and Theorem 3.3 of [13]

(i) Then M' is of the form $M' = \mathbb{L}_S(\!|\lambda x.M|\!)N$ and the proof is divided in two cases, depending on the substitution context $\mathbb{L}_S$. For instance, when $\mathbb{L}_S = (\!|\,\cdot\,|\!)$, then $M' = (\lambda x.M)N$ and the reduction occurs in the head, that is, $M' = (\lambda x.M)N \multimap_{\mathtt{dB}} M[N/x]$. Notice that

$$\begin{aligned}
[\![(\lambda x.M)N]\!]_a &= (\nu b)([\![\lambda x.M]\!]_b \mid (\nu y_0)b\langle y_0\rangle.((!y_0(x).[\![N]\!]_x) \mid [b \leftrightarrow a])) \\
&= (\nu b)(b(x).(\nu y_1)([x \leftrightarrow y_1] \mid [\![M]\!]_b\{y_1/u_x\}) \mid (\nu y_0)b\langle y_0\rangle.((!y_0(x).[\![N]\!]_x) \mid [b \leftrightarrow a])) \\
&\equiv (\nu b y_0)(b(x).(\nu y_1)([x \leftrightarrow y_1] \mid [\![M]\!]_b\{y_1/u_x\}) \mid b\langle y_0\rangle.((!y_0(x).[\![N]\!]_x) \mid [b \leftrightarrow a])) \\
&\Rightarrow_\tau (\nu b y_0)((\nu y_1)([y_0 \leftrightarrow y_1] \mid [\![M]\!]_b\{y_1/u_x\} \mid ((!y_0(x).[\![N]\!]_x) \mid [b \leftrightarrow a])) \\
&\Rightarrow_r^2 (\nu y_0)([\![M]\!]_a\{y_0/u_x\} \mid !y_0(x).[\![N]\!]_x) = [\![M[N/x]]\!]_a
\end{aligned}$$

The other cases can be found in the longer version of this paper [2].

□

Proposition 4.5 *There is no M such that $[\![M]\!]_a \Rightarrow_r Q$, for some Q.*

Proof. The proof follows by a direct inspection on the rule $\Rightarrow_r$. □

The next result adapts Lemma 7 in [1] to the proposed encoding and gives a characterisation of λ_{lsub}-terms whose encodings have some specific fragments, this will allow to work on simulation steps.

Lemma 4.6 *Let V and S a set of variable names and a set of special names, respectively.*

(i) *If $[\![M]\!]_a = \mathbb{N}_{V\uplus S}(\!|a(x).P|\!)$ with $a \notin S$ then $S = \emptyset$, $P = (\nu y)([x \leftrightarrow y]) \mid Q\{y/u_x\})$ and there exist N and $\mathbb{L}_V$ such that $Q = [\![N]\!]_a$ and $M = \mathbb{L}_V(\!|\lambda x.N|\!)$.*

(ii) *If $[\![M]\!]_a = \mathbb{N}_{V\uplus S}(\!|u_x\langle y\rangle.P|\!)$ with $y \in V$, and $u_x \notin V$ an unrestricted variable, then $P = [y \leftrightarrow a]$ and there exist $\Sigma \subseteq V$ and $\mathbb{E}_\Sigma$ such that $M = \mathbb{E}_\Sigma(\!|x|\!)$ (and $x \notin \Sigma$). Moreover, $\mathbb{N}_{V\uplus S} = (\nu y)(\mathbb{N}_{V'\uplus S})$ where $V = V' \uplus \{y\}$.*

Proof. In both cases the proof follows by induction on M. The full proof can be found in [2].

□

The theorem below shows that $\multimap$ simulates π but not so strongly as in [1], here we have a 1 for 3 relation, that is, one step of $\multimap$ corresponds to one step of $\Rightarrow_\tau$ followed by two renaming steps $\Rightarrow_r$.

Theorem 4.7 ($\multimap$ 3-simulates $\Rightarrow_\pi$ via $[\![]\!]_a$)

a) If $[\![M]\!]_a \Rightarrow_\tau Q$ then there exists N such that $M \multimap_{dB} N$ and $Q \Rightarrow_r^2 \equiv [\![N]\!]_a$.

b) If $[\![M]\!]_a \Rightarrow_{!\tau} Q$ then there exists N such that $M \multimap_{ls} N$ and $Q \Rightarrow_r \equiv [\![N]\!]_a$.

Proof. In both cases, the proof is by induction on the structure of M. The interesting case happens when $M = N[L/z]$ and the reduction occurs at the root. Then $[\![M]\!]_a = (\nu b)([\![N]\!]_a\{b/u_z\} \mid !b(c).[\![L]\!]_c)$ and we have to analyse the following cases:

a) The reduction $\Rightarrow_\tau$ cannot happen at the root.

b) In order to obtain such a reduction, $[\![N]\!]_a = \mathbb{N}_{V\uplus S}(\!|u_z\langle y\rangle.P|\!)$. By Lemma 4.6, $u_z \notin V$, $P = [y \leftrightarrow a]$ and $N = \mathbb{E}_\Sigma(\!|z|\!)$ for some context $\mathbb{E}_\Sigma$ and $\Sigma \subset V$. Besides, $\mathbb{N}_{V\uplus S} = (\nu y)(\mathbb{N}_{V'\uplus S})$, therefore,

$$\begin{aligned}
[\![M]\!]_a &= (\nu b)(\mathbb{N}_{V\uplus S}(\!|u_z\langle y\rangle.[y \leftrightarrow a]|\!)\{b/u_z\} \mid !b(c).[\![L]\!]_c) \\
&= (\nu b)((\nu y)\mathbb{N}_{V'\uplus S}(\!|u_z\langle y\rangle.[y \leftrightarrow a]|\!)\{b/u_z\} \mid !b(c).[\![L]\!]_c) \\
&\Rightarrow_{!\tau} (\nu b, y)(\mathbb{N}_{V'\uplus S}(\!|[y \leftrightarrow a] \mid [\![L]\!]_y \mid !b(c).[\![L]\!]_c|\!)) = Q
\end{aligned}$$

Notice that $M = N[L/z] = \mathbb{E}_\Sigma(\!|z|\!)[L/z] \multimap_{\mathtt{ls}} \mathbb{E}_\Sigma(\!|L|\!)[L/z]$. Also, $Q \Rightarrow_r (\nu b)(\mathbb{N}_{V'\uplus S}(\!|[\![L]\!]_a \mid !b(c).[\![L]\!]_c|\!)) = (\nu b)(\mathbb{N}_{V'\uplus S}(\!|[\![L]\!]_a\{b/u_z\} \mid !b(c).[\![L]\!]_c|\!)) = [\![L[L/z]]\!]_a$, since $u_z \notin var([\![L]\!]_a)$. The result follows by taking $N = \mathbb{E}_\Sigma(\!|L|\!)[L/z]$.

□

Remark 2 *Notice that we could obtain a 1-to-1 correspondence if we worked modulo renaming of names (modulo forwarding of names), say, via a relation $\equiv_r = \Leftrightarrow_r^*$.*

5 Conclusion and Ongoing Work

We have presented a type-directed encoding from the linear substitution λ-calculus (λ_{lsub}) to a fragment of synchronous π-calculus, combining the works in [13] and [1]. We have considered the linear weak head reduction semantics for the call-by-name evaluation of the λ_{lsub} and also a π-calculus at a distance. This expands the current theoretical investigations about the expressiveness of the π-calculus via encodings from the function-as-process paradigm. An ongoing work is the proposal of an encoding from λ_{lsub}-calculus to asynchronous π-calculus based on the translation proposed in [7] to session typed processes, where the 3-1 relation for simulation may be improved, while typed relations would still play a central role. Also, it would be interesting to develop encodings to strongly normalising λ-terms and to explore its connection with intersection types.

References

[1] B. Accattoli. Evaluating functions as processes. In *TERMGRAPH*, volume 110 of *EPTCS*, pages 41–55, 2013.

[2] S. Alves, D. Nantes-Sobrinho, J. A. Pérez, and D. Ventura. An investigation of linear substitution λ-calculus assession-typed processes. Technical report, 2020.

[3] H. P. Barendregt. *The lambda calculus - its syntax and semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1985.

[4] G. Boudol. The π-calculus in direct style. *High. Order Symb. Comput.*, 11(2):177–208, 1998.

[5] G. Boudol and C. Laneve. Lambda-calculus, multiplicities, and the pi-calculus. In *Proof, Language, and Interaction*, pages 659–690. The MIT Press, 2000.

[6] L. Caires and F. Pfenning. Session types as intuitionistic linear propositions. In P. Gastin and F. Laroussinie, editors, *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, volume 6269 of *Lecture Notes in Computer Science*, pages 222–236. Springer, 2010.

[7] H. DeYoung, L. Caires, F. Pfenning, and B. Toninho. Cut reduction in linear logic as asynchronous session-typed communication. In P. Cégielski and A. Durand, editors, *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, volume 16 of *LIPIcs*, pages 228–242. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[8] D. Kesner and D. Ventura. Quantitative types for the linear substitution calculus. In *IFIP TCS*, volume 8705 of *Lecture Notes in Computer Science*, pages 296–310. Springer, 2014.

[9] R. Milner. Functions as processes. *Math. Struct. Comput. Sci.*, 2(2):119–141, 1992.

[10] D. Sangiorgi. An investigation into functions as processes. In S. D. Brookes, M. G. Main, A. Melton, M. W. Mislove, and D. A. Schmidt, editors, *Mathematical Foundations of Programming Semantics, 9th International Conference, New Orleans, LA, USA, April 7-10, 1993, Proceedings*, volume 802 of *Lecture Notes in Computer Science*, pages 143–159. Springer, 1993.

[11] D. Sangiorgi. Lazy functions and mobile processes. Technical report, RR-2515, INRIA, 1995.

[12] D. Sangiorgi and X. Xu. Trees from functions as processes. *Logical Methods in Computer Science*, 14(3), 2018.

[13] B. Toninho, L. Caires, and F. Pfenning. Functions as session-typed processes. In *FoSSaCS*, volume 7213 of *Lecture Notes in Computer Science*, pages 346–360. Springer, 2012.