Implicative Logic based encoding of the λ-calculus into the π-calculus

Steffen van Bakel, Maria Grazia Vigliotti

Department of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ, UK

Abstract

We study an *output*-based encoding of the λ-calculus with explicit substitution into the synchronous π-calculus – enriched with pairing – that has its origin in mathematical logic, and show that this encoding respects reduction. We will define the notion of (explicit) spine reduction -which encompasses (explicit) lazy reduction- and show that the encoding fully encodes this reduction in that term-substitution as well as each single reduction step are modelled up to contextual similarity.

We show that all the main properties (soundness, completeness, and adequacy) hold for these four notions of reduction, as well as that termination is preserved.

We then define a notion of type assignment for the π-calculus that uses the type constructor $\rightarrow$, and show that all Curry types assignable to λ-terms are preserved by the encoding.

Key words: the λ-calculus, the π-calculus, intuitionistic logic, classical logic, encoding, type assignment

Introduction

In this paper we present a novel investigation of the encoding from various kind of the λ-calculi into the π-calculus. In particular, we are concerned with aspects of the encoding from the λ-calculus into the π-calculus such as: (1) interpreting λ-terms under *output* by explicitly naming the implicit *output* of terms; (2) modelling of a notion of explicit substitution into the π-calculus; (3) modelling more than just lazy reduction through applicative bi-simulation; (4) modelling individual steps in the reduction relation; (5) the definition a new logical notion of type assignment for the π-calculus; and (6) showing that implicative types for λ-terms can be assigned to their interpretation as well.

In the past, there have been several investigations of encoding from the λ-calculus into the π-calculus [19, 24]. Research in the direction of encodings of λ-terms was started by Milner in [19]; he defined an *input*-based encoding, and showed that the interpretation of closed λ-terms respects *lazy* reduction to normal form up to substitution. This approach has been picked up by many authors since then: it has been used by Sangiorgi [26], who also investigated it in the context of the higher-order π-calculus; by Honda *et al.* [17] with a rich type theory; and by Thielecke [28] in the context of continuation passing style programming languages, just to name a few. Milner also defined another *input*-based encoding that respects *call-by-value* reduction up to substitution, but this had fewer followers.

For many years, it seemed that the first and final word on the encoding of the λ-calculus has been said by Milner; in fact, Milner's encoding has set a milestone in the comparison between the functional and the concurrent paradigms, and all the above mentioned systems present variants of Milner's encoding[1].

In [24], Sangiorgi states good reasons for obtaining an expressive encoding: (1) from the process calculi point of view, to gain deeper insight into its theory; (2) from the λ-calculus point of view, to provide the opportunity to study λ-terms in contexts different than the sequential or applicative one; and (3) the λ-calculus is a model for functional language programming; these languages have never been very efficient, and one way of improving efficiency is to use parallel implementation.

Email addresses: s.vanbakel@imperial.ac.uk (Steffen van Bakel), maria.vigliotti@imperial.ac.uk (Maria Grazia Vigliotti)

[1] Sangiorgi [23] says: "*It seems established that* [Milner's encoding] *is canonical, i.e. it is the 'best' or 'simplest' encoding of the lazy λ-calculus into π-calculus.*"

We strongly agree with these assertions and we shall see in the course of this paper how our work on the λ-calculus with *explicit substitution* and a novel type system will shed new light on all these areas. We will choose an approach alternative to Milner's, interpreting terms under *output* rather than under *input*, and encoding spine reduction, which encompasses lazy reduction, in the process.

Substitution

The first question we wish to address here is:

> How can we faithfully model (implicit) substitution of the λ-calculus into the π-calculus.

This question is relevant, since Milner's encoding *does not* model implicit substitution (as illustrated in Figure 1).

There are two ways to address the problem of representing implicit λ-substitution in the π-calculus:

1. To encode the λ-calculus into the Higher-Order π-calculus, as in Sangiorgi [24];
2. To consider a different kind of λ-calculus where the substitution is more 'fine-grained'.

In this paper, we have chosen the second option, for which we need to understand what is exactly the notion of substitution that we can faithfully encode in π-calculus. Central to our approach is the interpretation of the *explicit substitution* version of reduction, which allows us to establish a clear connection between term-substitution in the λ-calculus, and the simulation of this operation in the π-calculus via channel-name passing.

Notice that, although both the λ-calculus and the π-calculus are equipped with substitution, these notions are conceptually very different. While the λ-calculus has an intrinsic '*high-order*' substitution mechanism by which terms get substituted for variables '*all in one go*', in the standard π-calculus the substitution mechanism replaces variables by channel names only, i.e. not by processes[2]. Because of this discrepancy, in the encoding of the λ-calculus into the π-calculus, it is not possible to assume that substitution in the former can be handled straightforwardly by the same mechanism in the latter.

We have to model the contraction of the redex $(\lambda x.M)N$ into the π-calculus, i.e. model the implicit substitution $M[N/x]$. Notice that the required number of copies needed of N is *a priori* unknown, so $M[N/x]$ has to be modelled using replication on N, as does $(\lambda x.M)N$. Since copies of replicated processes are extracted through the congruence $!P \equiv P \,|\, !P$, the replicated substitution will always be present during the computation of the process generated by the encoding. This is apparent in Milner's encoding [19] (see also the formulation of Milner's result Theorem 16, Theorem 18, Figure 1, and Example 22). However, terms of the substitution that are no longer needed (after all occurrences of x in M have been replaced) can be garbage-collected, so correctness of operational correspondence can be achieved with the help of strong bi-simulation.

This 'persistence' of the substitution in the π-calculus is absent in the standard λ-calculus, but is present in the our version of the (lazy) λ-calculus with explicit substitution, which is called the *explicit spine* calculus. Explicit substitution is generally considered an implementation of the λ-calculus, where every 'atomic step' of the substitution is represented in the reduction relation; in explicit spine reduction we restrict its applicability to head-variables only, leaving other occurrences untouched[3]. We will establish a correspondence between explicit spine reduction and synchronisation in the π-calculus, through which we emphasise that substitution in the π-calculus is more 'fine-grained' than that in λ-calculus, since it deals with substitutions '*one variable occurrence at a time*'.

We can then focus on what exactly is the notion of substitution that Milner's encoding *does* model, and shall argue that by encoding *explicit* rather than implicit substitution, Milner's encoding will enjoy a stronger operational correspondence (see Theorem 18), modelling individual *explicit lazy* reduction steps.

In what follows, we present two results: initially we 're-interpret' Milner's encoding into the *lazy explicit substitution λ-calculus*. We show that if we make substitution more 'fine-grained', we can obtain stronger operational correspondence results. We tease out, in our analysis, that substitution should be encoded directly – see Definition 17 – to obtain a more faithful encoding.

We will also present an encoding that is *conceptually different* from Milner's, being an interpretation under *output* rather than under *input*, which is a variant of the encoding presented in [7]. Another difference between our approach

[2] This is possible in the Higher-Order π-calculus [24].

[3] This is similar to reduction in Krivine's machine [18]

and Milner's is that we model abstraction via a process that uses asynchronous *output*; this will allow us to model reduction under λ-abstraction as well, i.e. our new encoding respects not only (explicit) lazy reduction, but also the (larger) notion of *explicit spine reduction*. Our encoding is not the first one to do that; in [21] the encoding of spine reduction λ-calculus into the Fusion-calculus is presented. To obtain their main result, in that paper there is no need to change Milner's encoding; that result is a consequence of the symmetric substitution mechanism introduced into the Fusion-calculus. Moreover, their encoding is not related to classical logic, as is ours.

For our encoding we do obtain a strong correspondence theorem between reductions, but we shall need the help of contextual equivalence to match terms perfectly. Furthermore, our encoding allows to establish a relationship between π-calculus and classical logic. In fact, the central idea behind this encoding interprets a redex (a logical cut) as a synchronisation, and is essentially based on Gentzen's encoding of Natural Deduction into the Sequent Calculus [12]. The idea of giving a computational interpretation of the cut as a synchronisation primitive is also used by [3] and [10]; in both papers, only a small fragment of Linear Logic was considered, and the encoding between proofs and the π-calculus was left rather implicit.

In summary, we feel we have gained faithfulness and clarity in considering the encoding from the λ-calculus with explicit substitution to the π-calculus. Our study shows that Milner's encoding, as well as ours, cannot fully represent implicit substitution, but does that for a limited version of explicit substitution, which, in a sense, is the minimal substitution needed to reach head-normal form.

Type system

We will show a type preservation result for our encoding, using the type system as presented in [4]. This type system is different from standard type systems for π as it does not contain any channel information and in that it expresses implication. It provides a logical view to the π-calculus, where π-processes can be witnesses of formulae that are provable (within the implicative fragment) in classical logic, as was shown in [4]; this implies that the π-calculus provides computational content to classical logic. This could suggest, in the long term, insights and advancement towards implementation of proof search for classical logic via abstract machines based on the π-calculus.

We will show in this paper that our encoding preserves types assignable to λ-terms in Curry's system. Through this type preservation result we show that our encoding also respects the *functional* behaviour of terms, as expressed via assignable types, and establish a stronger, deeper relationship between sequential/applicative and concurrent paradigms. Our work differs in spirit from the results by Honda, Yoshida and Berger [17] as their type system needs a linear restriction of the behaviour of the π-calculus to achieve a full abstraction result, as well as a typed language and a type-based interpretation.

The results on the type system that we present here determines the choice of the π-calculus used for the encoding: we use the synchronous π-calculus enriched with *pairs* of names [1]. In principle, our encoding could be adapted to the synchronous monadic π-calculus, however we would not be able to achieve the preservation of assignable types. Our encoding takes inspiration from, but it is a much improved version of, the encoding of λ-terms in to the sequent calculus $\mathcal{X}$ [5, 6] – a first variant was defined by Urban [29, 30]; $\mathcal{X}$ is a sequent calculus that enjoys the Curry-Howard correspondence for Gentzen's LK [13] – and the encoding of $\mathcal{X}$ into the π-calculus as defined in [4].

Our work not only sheds new light on the connection between functional and concurrent computation but also established a firm link between (classical) logic and process calculi, as first reported on in [4], a very promising area of research.

In summary, the main achievements of this paper are:

- Reinterpretation of Milner's encoding of λ-calculus with explicit substitution into π-calculus with strong operational correspondence theorem.
- An *output*-based encoding of the λ-calculus with explicit substitution into the synchronous π-calculus with pairing is defined that preserves spine reduction for all terms up to contextual equivalence, and, by inclusion, for lazy reduction with explicit substitution;
- The encoding respects implicit substitution, and respects both spine reduction and lazy reduction for closed terms up to simulation;

- The encoding preserves assignable Curry types for λ-terms, with respect to the context assignment system for π from [4].

Paper outline

In Section 1, we repeat the definition of the synchronous π-calculus with pairing, and in Section 2 that of the λ-calculus, where we present the notion of explicit spine reduction '$\rightarrow_{\mathbf{xS}}$' which takes a central role in this paper; in Section 3 we also briefly discuss Milner's interpretation result for the lazy λ-calculus, as well as Sangiorgi and Walker's *uniform* encoding [26]. Then, in Section 4, we will define an encoding where terms are interpreted under *output* rather than *input* (as in Milner's), and show in Section 5 that $\rightarrow_{\mathbf{xS}}$ is respected by our interpretation, modulo renaming. In Section 6, we show that this renaming is not needed when interpreting the lazy λ-calculus. To conclude, in Section 7 we give a notion of (type) context assignment on processes in π, and show that our interpretation preserves types. In fact, this result is the main motivation for our interpretation, which is therefore *logical.*

This paper is a modified version of the paper that appeared as [7]; we have, in particular, addressed the termination issue.

1. The synchronous π-calculus with pairing

The notion of synchronous π-calculus that we consider in this paper is similar to the one used also in [1, 4], and is different from other systems studied in the literature [15] in a number of aspects: we add pairing, and introduce the *let*-construct to deal with inputs of pairs of names that get distributed. The main reason for the addition of pairing [1] lies in the fact that we want to preserve implicate type assignment.

The π-calculus is an *input-output* calculus, where terms have not just more than one *input,* but also more than one *output.* This is similar to what we find in Gentzen's LK, where right-introduction of the arrow is represented by

$$(\Rightarrow R): \quad \frac{\Gamma, A \vdash_{\mathrm{LK}} B, \Delta}{\Gamma \vdash_{\mathrm{LK}} A \Rightarrow B, \Delta}$$

with Γ and Δ multi-sets of formulae. Notice that only *one* of the possible formulae is selected from the right context, and *two* formulae are selected in *one* step; when searching for a Curry-Howard correspondence, this will have to be reflected in the (syntactic) witness of the proof[4]. Now if we want to model this in π, i.e. want to express function construction (abstraction), we would also need to bind *two* free names, one as name for the *input* of the function, and the other as name for its *output.* We can thus express that a process P acts as a function *only* when fixing (binding) *both* an *input and* an *output simultaneously*, i.e. in *one* step; we use pairing exactly for this: interfaces for functions are modelled by sending and receiving *pairs* of names.

We will introduce *data* as a structure over names, such that not only names but also pairs of names can be sent (but not a pair of pairs); this way a channel may pass along either a name or a pair of names.

Definition 1 (Processes). *Channel names* and *data* are defined by:

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

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

$$\begin{array}{rcll} P, Q & ::= & 0 & \textit{Nil} \\ & \mid & P \mid Q & \textit{Composition} \\ & \mid & !\,P & \textit{Replication} \\ & \mid & (\nu a)\,P & \textit{Restriction} \\ & \mid & a(x).\,P & \textit{Input} \\ & \mid & \overline{a}\langle p\rangle.\,P & \textit{Output} \\ & \mid & \textit{let}\,\langle x, y\rangle = p\ \textit{in}\ P & \textit{Let construct} \end{array}$$

We see, as usual, ν as a binder, and call the name n *bound* in $(\nu n)\,P$; n is *free* in P if it occurs in P, but is not bound.

We call a variable/name *visible* in P if it occurs free, and does not occur under an *input* or an *output.* A *context* $\mathsf{C}[\cdot]$ is a process with a hole $[\,]$.

[4] This is exactly the approach of the calculus $\mathcal{X}$, where the representative of $(\Rightarrow R)$ binds two connectors.

We abbreviate $a(x).\,let\,\langle y,z\rangle = x\ in\ P$ by $a(y,z).\,P$, and $(\nu m)(\nu n)\,P$ by $(\nu mn)\,P$, and write $\overline{a}\langle p\rangle$ for $\overline{a}\langle p\rangle.\,\mathbf{0}$, and $\overline{a}\langle c,d\rangle.\,P$ for $\overline{a}\langle\langle c,d\rangle\rangle.\,P$. A (process) context is simply a term with a hole $[\cdot]$.

Definition 2 (Congruence). The structural congruence is the smallest equivalence relation closed under contexts defined by the following rules:

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

We will consider processes modulo congruence: this implies that we will not deal explicitly with the process $let\,\langle x,y\rangle = \langle a,b\rangle\ in\ P$, but rather with $P[a/x,b/y]$.

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

Definition 3 (Reduction). The *reduction relation* is defined by the following (elementary) rules:

$$\begin{array}{rcll}
\overline{a}\langle p\rangle.\,P \mid a(x).\,Q & \to_\pi & P \mid Q[p/x] & (\textit{synchronisation}) \\
P \to_\pi P' & \Rightarrow & (\nu n)\,P \to_\pi (\nu n)\,P' & (\textit{hiding}) \\
P \to_\pi P' & \Rightarrow & P \mid Q \to_\pi P' \mid Q & (\textit{composition}) \\
P \equiv Q \ \&\ Q \to_\pi Q' \ \&\ Q' \equiv P' & \Rightarrow & P \to_\pi P' & (\textit{congruence})
\end{array}$$

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

We will use the notation $\to^+$ and $\to^*$ for all notions of reduction we discuss here.

Notice that

$$\begin{array}{rcll}
\overline{a}\langle b,c\rangle.\,P \mid a(x,y).\,Q & \triangleq & \overline{a}\langle\langle b,c\rangle\rangle.\,P \mid a(z).\,let\,\langle x,y\rangle = z\ in\ Q & \to_\pi \\
 & & P \mid let\,\langle x,y\rangle = \langle b,c\rangle\ in\ Q & \equiv\quad P \mid Q[b/x,c/y].
\end{array}$$

Definition 4.

1. We write $P \downarrow n$ (and say that P *outputs 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 $P \Downarrow n$ (and say that P *will output on* n) if there exists Q such that $P \to_\pi^* Q$ and $Q \downarrow n$.
3. We write $P \sim_{\mathrm{C}} Q$ (and call P and Q *contextually equivalent*) if, for all contexts $\mathsf{C}[\cdot]$, and for all n, $\mathsf{C}[P] \Downarrow n$ if and only if $\mathsf{C}[Q] \Downarrow n$.

2. The Lambda Calculus (and variants thereof)

We assume the reader to be familiar with the λ-calculus; we just repeat the definition of the relevant notions.

Definition 5 (Lambda terms and β-contraction [8]).

1. The set Λ of *λ-terms* is defined by the grammar:

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

2. The one-step reduction relation $\to_\beta$ is defined by the rules:

$$(\beta): \quad (\lambda x.M)N \ \to\ M[N/x] \qquad M \to N \ \Rightarrow\ \begin{cases} ML \ \to\ NL \\ LM \ \to\ LN \\ \lambda x.M \ \to\ \lambda x.N \end{cases}$$

where $M[N/x]$ is the (implicit) substitution of N for x in M, which takes place immediately and silently.

We will focus in this paper mainly on Call-By-Name reduction systems, in the sense that, in an application MN, reduction will take place only in M until it either (1) terminates in an abstraction $\lambda x.P$, after which we will contract the redex $(\lambda x.P)N$, or (2) it will terminate when reaching a variable. The two main notions are *lazy* reduction, where reduction stops on M when an abstraction is created, and *spine* reduction, where we also can contract (head) redexes inside an abstraction. Since these notions are defined by limiting the *contextual* reduction rules of the λ-calculus, in all notions we present here those rules are present (as above); this is in contrast to normal presentations that leave the contextual rules implicit. Moreover, in view of the fact that we aim to build encodings of these notions of reduction into the π-calculus where we encoding of normal reduction is intricate, we will consider versions of those two notions with *explicit substitution*, that can be accurately encoded.

How to deal with implicit substitution $[N/x]$ on terms plays an important role in interpretations/implementations of the λ-calculus. To encode β-reduction $(\lambda x.M)N \to_\beta M[N/x]$ in the π-calculus, implicit substitution has to be modelled using synchronisation, since this is the only computational action in the π-calculus. However, remark that synchronisation takes place one-at-the-time (i.e. *one output* synchronising with *one input*); since a priori the required number of copies needed of N is unknown, the distributive character of the substitution of N for x in M has to be modelled using replication. Also, the interpretation of $M[N/x]$ itself is the result of running the interpretation of $(\lambda x.M)N$; since no step in π introduces replication, it is clear that also in the latter, the interpretation of N must appear replicated in the same way.

As is clear from the formulation of Milner's result (see Theorem 16), since $!P \equiv P \mid !P$, even when all xs in M have disappeared as result of the execution of the interpretation of the substitution $M[N/x]$, the replicated substitution term will always remain. To not generate too many running copies of N than are strictly needed, Milner engineered his encoding to block the running of $[N/x]$ by placing an *output* guard (as in $!\,x(w).[\![N]\!]^{\mathrm{M}}\,w$), making the synchronisation over x the deblocking action. Since the definition of reduction on the π-calculus does not permit synchronisation under replication or guard, this implies that reductions in the right-hand term of an application cannot be modelled. Also, since λ-abstraction is modelled by Milner via *input*, reduction under an abstraction cannot be modelled. These two restrictions imply that, using Milner's approach, resulting in his encoding of the λ-calculus into the (synchronous, monadic) π-calculus as defined in Definition 13, only the *lazy* λ-calculus can be modelled, as defined below.

We also define *spine* reduction, which we will encode in this paper.

Definition 6 (Lazy and spine reduction). 1. *Lazy* reduction[5] for the λ-calculus [2] is defined by limiting the one-step reduction relation to:

$$(\lambda x.M)N \;\to\; M[N/x] \qquad M \to N \;\Rightarrow\; ML \to NL$$

We write $P \to_{\mathrm{L}} Q$ if P reduces to Q using lazy reduction.

2. We define *spine* reduction[6] by limiting one-step reduction to:

$$(\lambda x.M)N \;\to\; M[N/x] \qquad M \to N \;\Rightarrow\; \begin{cases} ML \;\to\; NL \\ \lambda x.M \;\to\; \lambda x.N \end{cases}$$

We write $M \to_{\mathrm{S}} N$ if M reduces to N using spine reduction.

Notice that spine reduction is aptly named, since all reductions take place on the spine of the λ-tree (see [9]): searching for a redex, starting from the root, we can walk 'down' and turn 'left', but not turn 'right', so stay on the spine of the tree. This notion of reduction is shown to be head-normalising in [9] (even quasi-head normalising); in fact, the normal forms for spine reduction are exactly the head-normal forms for normal reduction [31].

[5]This reduction relation is sometimes also known as 'Call-by-Name'; since this is an overloaded concept, we stick to the terminology 'lazy'; the definition here is the one used in [19].

[6]This notion is known also as 'strong Call-by-Name'; in [14], essentially following [9], spine reduction is defined by "*just contracting redexes that are on the spine*"; *head* spine reduction is mentioned, but not defined, in [27].

Notice that spine reduction encompasses lazy reduction, since $M \to_{\mathrm{L}} N$ implies $M \to_{\mathrm{S}} N$, but not vice-versa, since both

$$(\lambda x.(\lambda y.M)N)L \quad \to_{\mathrm{S}} \quad \begin{cases} (\lambda x.M[N/y])L \\ ((\lambda y.M)N)[L/x] \end{cases}$$

whereas only $(\lambda x.(\lambda y.M)N)L \to_{\mathrm{L}} ((\lambda y.M)N)[L/x]$.

It is worthwhile to note that, although not mentioned in [19], the proof of Milner's main result (Theorem 16 in this paper) treats the substitution as *explicit*, not as *implicit*; for example, in the proof of Lemma 4.5 in that paper, case 3 considers the term $xM_1 \cdots M_n[N/x]$ and $NM_1 \cdots M_n$ to be *different*. It is therefore opportune to switch our attention to Bloo and Rose's calculus $\lambda\mathbf{x}$ [11], a calculus with explicit substitution, where a β-reduction of the λ-calculus is split into several more atomic steps of computation[7]. Bloo and Rose add the concept of substitution to the syntax of the calculus, making it *explicit*, by adding the operator $M\langle x := N\rangle$:

The syntax of the *explicit λ-calculus* $\lambda\mathbf{x}$ is an extension of that of the λ-calculus.

Definition 7 (Syntax of λx c.f. [11]). $M, N \quad ::= \quad x \mid \lambda x.M \mid MN \mid M\langle x := N\rangle$

A term M is called *pure* if it contains no explicit substitution $\langle x := N\rangle$.

We write $MN\langle x := L\rangle$ for $(MN)\langle x := L\rangle$, and $M\langle y := N\rangle\langle x := L\rangle$ for $(M\langle y := N\rangle)\langle x := L\rangle$, and use $M\,\overrightarrow{\langle y := L\rangle}$ as shorthand for $M\langle y_1 := L_1\rangle \cdots \langle y_n := L_n\rangle$, with $n \geq 0$; by Barendregt's convention, all y_i are distinct.

Explicit substitution describes explicitly the process of executing a β-reduction, i.e. expresses syntactically the details of the computation as a succession of atomic, constant-time steps (in a first-order rewriting system), where the implicit substitution of the β-reduction step is split into several steps.

Definition 8 (Reduction on λx). The reduction relation $\to_{\mathbf{x}}$ on terms in $\lambda\mathbf{x}$ is defined by the following rules[8]:

$$\begin{array}{rrcll} (\mathsf{B}): & (\lambda x.M)P & \to & M\langle x := P\rangle & \\ (\mathsf{App}): & (MN)\langle x := P\rangle & \to & M\langle x := P\rangle N\langle x := P\rangle & \\ (\mathsf{Abs}): & (\lambda y.M)\langle x := P\rangle & \to & \lambda y.(M\langle x := P\rangle) & \\ (\mathsf{VarI}): & x\langle x := P\rangle & \to & P & \\ (\mathsf{gc}): & M\langle x := P\rangle & \to & M, \qquad x \notin \mathit{fv}(M) & \end{array} \qquad M \to N \;\Rightarrow\; \begin{cases} ML & \to & NL \\ LM & \to & LN \\ \lambda x.M & \to & \lambda x.N \\ M\langle x := L\rangle & \to & N\langle x := L\rangle \\ L\langle x := M\rangle & \to & L\langle x := N\rangle \end{cases}$$

We write $\to_{:=}$ if only the rules (App), (Abs), (VarI), and (VarK) are applied in the reduction.

We observe that $\to_{:=}$ implements the implicit substitution of the λ-calculus; notice that reductions in $\to_{:=}$ terminate.

Although stated with implicit substitution, Milner's result (Theorem 16) does not show that lazy reduction is fully modelled, as can be observed in Figure 1; rather, it models applicative bi-simulation only. Although in the proof of his result Milner treats substitution as explicit, careful analysis shows that even $\lambda\mathbf{x}$'s reduction is not fully modelled by Milner's encoding; however, explicit lazy reduction – a more restricted version, that we will define below – is.

We will, in fact, distinguish *two* notions and define also explicit spine reduction; we will show that step-by-step reduction in the first is modelled by Milner's encoding, and that step-by-step reduction in the second is modelled by our encoding (see Definition 25), up to renaming.

Definition 9 ($\lambda\mathbf{x}_{\mathrm{L}}$ and $\lambda\mathbf{x}_{\mathrm{S}}$).

1. The syntax of the *explicit lazy λ-calculus* $\lambda\mathbf{x}_{\mathrm{L}}$ and that of the *explicit spine λ-calculus* $\lambda\mathbf{x}_{\mathrm{S}}$ is that of $\lambda\mathbf{x}$.
2. The explicit variant $\to_{\mathbf{x}_{\mathrm{L}}}$ of lazy reduction is defined as follows.

$$\begin{array}{rcl} (\lambda x.M)N & \to & M\langle x := N\rangle \\ xM_1 \cdots M_n\,\overrightarrow{\langle y := L\rangle}\langle x := N\rangle & \to & NM_1 \cdots M_n\,\overrightarrow{\langle y := L\rangle}\langle x := N\rangle \end{array}$$

$$M \to N \;\Rightarrow\; \begin{cases} ML & \to & NL \\ M\langle x := L\rangle & \to & N\langle x := L\rangle \end{cases}$$

[7] Many other notions of calculi with explicit substitution exist, but those are not relevant to our results.

[8] An alternative to the fifth rule is the rule (VarK), defined by $y\langle x := P\rangle \to y$.

3. *Explicit spine reduction* $\to_{\mathbf{xs}}$ is defined via:

$$\begin{array}{rcl}(\lambda x.M)N & \to & M\langle x := N\rangle \\ (\lambda y.M)\langle x := N\rangle & \to & \lambda y.(M\langle x := N\rangle) \\ xM_1\cdots M_n\,\overrightarrow{\langle y := L\rangle}\langle x := N\rangle & \to & NM_1\cdots M_n\,\overrightarrow{\langle y := L\rangle}\langle x := N\rangle\end{array}$$

$$M \to N \;\Rightarrow\; \begin{cases} ML & \to NL \\ \lambda x.M & \to \lambda x.N \\ M\langle x := L\rangle & \to N\langle x := L\rangle \end{cases}$$

4. We call x the *lazy head variable* of $xM_1\cdots M_n\,\langle\overrightarrow{y := N}\rangle$, and the *head variable* of $\lambda\vec{z}.xM_1\cdots M_n\,\langle\overrightarrow{y := N}\rangle$.

Notice that we deviate here from the approach of $\lambda\mathbf{x}$ by using a notion of explicit substitution that is *lazy*, i.e. we postpone substitutions until the (head-)reduction has reached the stage that the term to be substituted is needed in order to be able to continue with the reduction. Remark that, in the context of *implicit* substitution, we have no choice but to accept that, when contracting a redex $(\lambda x.M)N$, the parameter N immediately gets substituted for *all* the occurrences of x in M. When moving to the context of *explicit* substitution, this is no longer the case, and we can gain control over exactly which occurrences of x do effectively need to be replaced immediately, and which can be postponed until a later moment. We will see that this behaviour corresponds directly to the behaviour of the encoded terms in the π-calculus.

The criterion, in the context of lazy reduction, is of course to perform only those substitutions that are essential for the continuation of reduction: for example, when contracting $(xx)\langle x := \lambda y.y\rangle$, only the substitution to the head variable is essential to make sure that lazy reduction can continue: this would yield $((\lambda y.y)x)\langle x := \lambda y.y\rangle$. The second x will only be replaced when it becomes the head-variable[9], i.e. after the redex $(\lambda y.y)x$ gets contracted, yielding $y\langle y := x\rangle\langle x := \lambda y.y\rangle$, which in turn reduces to $x\langle y := x\rangle\langle x := \lambda y.y\rangle$; now the variable is at the head, the postponed substitution can be applied which in turn yields $(\lambda z.z)\langle y := x\rangle\langle x := \lambda y.y\rangle$ (notice that this reduction is not in $\lambda\mathbf{x}$). So, in general, lazy explicit substitution replaces only the lazy head variable of a term.

Explicit lazy reduction of $\lambda\mathbf{x}_{\mathrm{L}}$ has similarities with Krivine's machine [18], since the explicit substitutions correspond to *closures*. Krivine's machine is deterministic and stops at weak-head normal from, i.e. does not reduce under an abstraction, as in the explicit lazy reduction: this is not true for explicit spine reduction. Krivine's machine therefore corresponds more to explicit lazy reduction.

The following is easy to show:

Proposition 10. *If $M \to_{\mathbf{xs}} N$, then there exists a pure λ-term L such that $N \to_{:=} L$.*

Since spine reduction reduces a term M to head-normal form, if it exists, this implies that also $\to_{\mathbf{xs}}$ reduces to head-normal form, albeit with perhaps some substitutions still pending.

Example 11. 1. Substitutions are left after reducing, like $(\lambda z.yz)N \to_{\mathbf{xs}} yz\langle z := N\rangle$.

We can reduce $(\lambda x.(Lz.A(Ly.M)Vx))N$ in two different ways:

$$\begin{array}{lll}(\lambda x.(\lambda z.(\lambda y.M)x))N & \to_{\mathbf{xs}} & (\lambda x.(\lambda z.(\lambda y.M)x))N \quad \to_{\mathbf{xs}} \\ (\lambda z.(\lambda y.M)x)\langle x := N\rangle & \to_{\mathbf{xs}} & (\lambda x.(\lambda z.(M\langle y := x\rangle)))N \quad \to_{\mathbf{xs}} \\ \lambda z.((\lambda y.M)x)\langle x := N\rangle & \to_{\mathbf{xs}} & \lambda z.(M\langle y := x\rangle)\langle x := N\rangle \quad \to_{\mathbf{xs}} \\ \lambda z.(M\langle y := x\rangle\langle x := N\rangle) & & \lambda z.(M\langle y := x\rangle\langle x := N\rangle)\end{array}$$

Notice that

$$\begin{array}{lll}(\lambda x.xx)(\lambda y.y) \to_{\mathbf{xs}} & xx\langle x := \lambda y.y\rangle & \to_{\mathbf{xs}} \\ & (\lambda y.y)x\langle x := \lambda y.y\rangle & \to_{\mathbf{xs}} \\ & y\langle y := x\rangle\langle x := \lambda y.y\rangle & \to_{\mathbf{xs}} \\ & x\langle y := x\rangle\langle x := \lambda y.y\rangle & \to_{\mathbf{xs}} (\alpha) \quad (\lambda z.z)\langle y := x\rangle\langle x := \lambda y.y\rangle\end{array}$$

[9] This appears to be the implicit approach of [19] (see Lemma 4.5, case 3).

2. Of course in $\lambda\mathbf{x}$S we can have non-terminating reductions, as illustrated by:

$$\begin{array}{lll|lll}
(\lambda x.xx)(\lambda x.xx) & \to_{\mathbf{x}\mathrm{S}} & & & & \\
xx\langle x := \lambda y.yy\rangle & \to_{\mathbf{x}\mathrm{S}} & & zz\langle z := y\rangle\langle y := x\rangle\langle x := \lambda y.yy\rangle & \to_{:=}^{+} & \\
(\lambda y.yy)x\langle x := \lambda y.yy\rangle & \to_{\mathbf{x}\mathrm{S}} & & yy\langle y := x\rangle\langle x := \lambda y.yy\rangle & \to_{:=}^{+} & \\
yy\langle y := x\rangle\langle x := \lambda y.yy\rangle & \to_{\mathbf{x}\mathrm{S}} & & xx\langle x := \lambda y.yy\rangle & \to_{:=}^{+} & \\
xy\langle y := x\rangle\langle x := \lambda y.yy\rangle & \to_{\mathbf{x}\mathrm{S}} & (\alpha) & (\lambda y.yy)(\lambda y.yy) & & \\
(\lambda z.zz)y\langle y := x\rangle\langle x := \lambda y.yy\rangle & \to_{\mathbf{x}\mathrm{S}} & & & & \\
zz\langle z := y\rangle\langle y := x\rangle\langle x := \lambda y.yy\rangle & \to_{\mathbf{x}\mathrm{S}}^{+} & \cdots & & &
\end{array}$$

(notice the α-conversion, needed to adhere to Barendregt's convention). This reduction is deterministic and clearly loops; however, notice that $(\lambda x.xx)(\lambda x.xx)$ does not run to itself. The second part shows that, as stated by Proposition 12, the normal reduction result can be achieved by reduction in $\lambda\mathbf{x}$.

We can easily show the following result, that states the relation between the various notions of reduction:

Proposition 12. 1. *If $M \to_{\mathrm{L}}^{*} N$, then there exists $L \in \lambda\mathbf{x}$ such that $M \to_{\mathbf{x}\mathrm{L}}^{*} L$ and $L \to_{:=}^{*} N$.*
2. *If $M \to_{\mathrm{L}}^{*} N$, with N in normal form, then there exists $L \in \lambda\mathbf{x}$ such that L is in $\lambda\mathbf{x}$L-normal form, and $M \to_{\mathbf{x}\mathrm{L}}^{*} L$ and $L \to_{:=}^{*} N$.*
3. *If $M \to_{\mathrm{S}}^{*} N$, then there exists $P, \vec{x}, \vec{Q}$ such that $M \to_{\mathbf{x}\mathrm{S}}^{*} P\langle\overrightarrow{x := Q}\rangle \to_{:=}^{*} N$, and $P[\overrightarrow{Q/x}] = N$.*
4. *If $M \to_{\mathrm{S}}^{*} N$, and N is in $\to_{\mathrm{S}}$-normal form (i.e. in β-head-normal form), then there exists $L \in \lambda\mathbf{x}$ such that L is in $\lambda\mathbf{x}$S-normal form, and $M \to_{\mathbf{x}\mathrm{S}}^{*} L$ and $L \to_{:=}^{*} N$.*

Notice that, in particular, the second part holds for *single step* reductions.

3. Milner's *input*-based lazy encoding

In his seminal paper [19], Milner defines an encoding of the λ-calculus into the (monadic) π-calculus, and shows some correctness results. His Call-By-Name encoding[10] is inspired by the normal semantics of λ-terms, which states for abstraction:

$$[\![\lambda x.M]\!]_{\xi}^{\mathcal{M}} \quad = \quad G(\lambda\!\!\lambda d \in \mathcal{M}.[\![M]\!]_{\xi(d/x)}^{\mathcal{M}})$$

(here $\mathcal{M}$ is an domain, ξ is a *valuation*, mapping free variables to elements of the domain, G a domain constructor, and the body of the abstraction is interpreted in the updated valuation, where now also x is mapped to d, an arbitrary element of the domain[11]). So, also in the encoding, instead of executing $M[N/x]$, M is executed in an environment that binds N to the variable x. This leads to:

Definition 13 (Milner's interpretation [19]). The *input-based encoding* of λ-terms into the π-calculus is defined by:

$$\begin{array}{rcll}
[\![x]\!]^{\mathrm{M}}\, a & \triangleq & \overline{x}\langle a\rangle & x \neq a \\
[\![\lambda x.M]\!]^{\mathrm{M}}\, a & \triangleq & a(x).\, a(b).\, [\![M]\!]^{\mathrm{M}}\, b & b \textit{ fresh} \\
[\![MN]\!]^{\mathrm{M}}\, a & \triangleq & (\nu c)\,([\![M]\!]^{\mathrm{M}}\, c \mid (\nu z)\,(\overline{c}\langle z\rangle.\, \overline{c}\langle a\rangle.\, [\![z := N]\!]^{\mathrm{M}})) & c, z \textit{ fresh} \\
[\![x := M]\!]^{\mathrm{M}} & \triangleq & !\, x(w).\, [\![M]\!]^{\mathrm{M}}\, w & w \textit{ fresh}
\end{array}$$

Milner calls $[\![x := M]\!]^{\mathrm{M}}$ an "*environment entry*"; it could be omitted from the definition above, but is of use separately. Here a is the channel along which $[\![M]\!]^{\mathrm{M}}$ receives its argument; this is used to communicate with the interpretation of the argument, as made clear in the third case, were the *input* channel of the left-hand term is used to send the name over on which the right-hand term will receive its *input.*

[10] [19] also deals with Call-By-Value, which we will not consider here.
[11] For details, see [8].

We would, at this stage, like to draw attention to the 'delay' mechanism that forms part of this encoding through the guard on the substitution $!\,z(w).\,[\![N]\!]^{\mathrm{M}}\,w$. This has not only the advantage that reduction cannot take place 'in the argument of substitution' $[\![N]\!]^{\mathrm{M}}\,w$, also, no context can interact with this term; even after a copy has been peeled of from the replicated term (via $!\,x(w).\,[\![N]\!]^{\mathrm{M}}\,w \equiv x(w).\,[\![N]\!]^{\mathrm{M}}\,w \mid !\,x(w).\,[\![N]\!]^{\mathrm{M}}\,w$) the process $[\![N]\!]^{\mathrm{M}}\,x$ cannot start running, since guarded by an *input.* This is made evident in the next example.

Example 14.

$$
\begin{array}{lll}
[\![(\lambda x.x)N]\!]^{\mathrm{M}}\,a & \triangleq & \\
(\nu c)\,(c(x).\,c(b).\,\overline{x}\langle b\rangle \mid (\nu z)\,(\overline{c}\langle z\rangle.\,\overline{c}\langle a\rangle.\,[\![z := N]\!]^{\mathrm{M}})) & \to_{\pi} & (c) \\
(\nu z)\,(\overline{z}\langle a\rangle \mid !\,z(w).\,[\![N]\!]^{\mathrm{M}}\,w) & \equiv & \\
(\nu z)\,(\overline{z}\langle a\rangle \mid z(w).\,[\![N]\!]^{\mathrm{M}}\,w \mid !\,z(w).\,[\![N]\!]^{\mathrm{M}}\,w) & \to_{\pi} & (z) \\
(\nu z)\,([\![N]\!]^{\mathrm{M}}\,a \mid !\,z(w).\,[\![N]\!]^{\mathrm{M}}\,w) & \equiv & \\
[\![N]\!]^{\mathrm{M}}\,a \mid (\nu z)\,(!\,z(w).\,[\![N]\!]^{\mathrm{M}}\,w) & \sim_{\mathrm{C}} & [\![N]\!]^{\mathrm{M}}\,a
\end{array}
$$

Notice that $[\![N]\!]^{\mathrm{M}}\,w$ cannot run until the synchronisation over z has taken place, and that $(\nu z)\,(!\,z(w).\,[\![N]\!]^{\mathrm{M}}\,w)$ is contextually equivalent to $\mathbf{0}$.

Milner's initial approach has since become standard; in fact, as mentioned in the introduction, Sangiorgi considers it canonical [23].

Notice that both the body of the abstraction and the argument in an application both get positioned under an *input*, and that therefore reductions inside these subterms cannot be modelled; as a result, the simulation via the encoding is limited to lazy reduction.

Example 15. Using $[\![\cdot]\!]^{\mathrm{M}}\,\cdot$, the interpretation of a β-redex (only) reduces as follows:

$$
\begin{array}{lll}
[\![(\lambda x.M)N]\!]^{\mathrm{M}}\,a & \triangleq & \\
(\nu c)\,([\![\lambda x.M]\!]^{\mathrm{M}}\,c \mid (\nu z)\,(\overline{c}\langle z\rangle.\,\overline{c}\langle a\rangle.\,[\![z := N]\!]^{\mathrm{M}})) & \triangleq & \\
(\nu c)\,(c(x).\,c(b).\,[\![M]\!]^{\mathrm{M}}\,b \mid (\nu z)\,(\overline{c}\langle z\rangle.\,\overline{c}\langle a\rangle.\,[\![z := N]\!]^{\mathrm{M}})) & \to_{\pi} & (c) \\
(\nu cz)\,(c(b).\,[\![M[z/x]]\!]^{\mathrm{M}}\,b \mid \overline{c}\langle a\rangle.\,[\![z := N]\!]^{\mathrm{M}}) & \to_{\pi} & (c) \\
(\nu z)\,([\![M[z/x]]\!]^{\mathrm{M}}\,a \mid [\![z := N]\!]^{\mathrm{M}}) & =_{\alpha} & (z \notin [\![M]\!]^{\mathrm{M}}\,b) \\
(\nu x)\,([\![M]\!]^{\mathrm{M}}\,a \mid [\![x := N]\!]^{\mathrm{M}}) & \triangleq & \\
(\nu x)\,([\![M]\!]^{\mathrm{M}}\,a \mid !\,x(w).\,[\![N]\!]^{\mathrm{M}}\,w) & &
\end{array}
$$

Now reduction can continue in (the interpretation of) M, but not in N that is still guarded by the *input* on x, which will not be used until the evaluation of $[\![M]\!]^{\mathrm{M}}\,a$ reaches the point where *output* is generated over x.

Milner shows the main correctness result for his interpretation with respect to lazy reduction, which is stated as follows:

Theorem 16 ([19]). *For all closed λ-terms M, either:*

1. $M \to_{\mathrm{L}}^{*} \lambda y.R[\overrightarrow{N/x}]$, *and* $[\![M]\!]^{\mathrm{M}}\,u \to_{\pi}^{*} (\overrightarrow{\nu x})\,([\![\lambda y.R]\!]^{\mathrm{M}}\,u \mid [\![\overrightarrow{x := N}]\!]^{\mathrm{M}})$, *or*
2. *both M and $[\![M]\!]^{\mathrm{M}}\,u$ diverge.*

We have already remarked that Milner's encoding does not simulate step-by-step reduction, not even when restricted to lazy reduction. To illustrate this fact, Figure 1 shows the reduction of the interpretation of $(\lambda x.xx)(\lambda y.y)$ under Milner's encoding. Notice that, there, we have executed the only possible synchronisations, and that in the reduction path no term corresponds to

$$(\nu c)\,([\![\lambda y.y]\!]^{\mathrm{M}}\,c \mid (\nu z)\,(\overline{c}\langle z\rangle.\,\overline{c}\langle a\rangle.\,[\![z := \lambda y.y]\!]^{\mathrm{M}})) \quad = \quad [\![(\lambda y.y)(\lambda y.y)]\!]^{\mathrm{M}}\,a$$

So, in particular, $(\nu z)\,([\![zz]\!]^{\mathrm{M}}\,a \mid [\![z := \lambda y.y]\!]^{\mathrm{M}})$ does *not* reduce to the result of the substitution, $[\![(\lambda y.y)(\lambda y.y)]\!]^{\mathrm{M}}\,a$: Milner's encoding does not model implicit substitutions itself. Therefore, although $(\nu x)\,([\![M]\!]^{\mathrm{M}}\,a \mid [\![x := N]\!]^{\mathrm{M}})$ can intuitively be seen as $[\![M[N/x]]\!]^{\mathrm{M}}\,a$, these are, in fact, substantially different.

$$
\begin{array}{lll}
& [\![(\lambda x.xx)(\lambda y.y)]\!]^{\mathrm{M}}\, a & \triangleq \\
1: & (\nu c)\,([\![\lambda x.xx]\!]^{\mathrm{M}}\, c \mid (\nu z)\,(\overline{c}\langle z\rangle.\,\overline{c}\langle a\rangle.\,[\![z := \lambda y.y]\!]^{\mathrm{M}})) & \triangleq \\
& (\nu c)\,(c(x).\,c(b).\,[\![xx]\!]^{\mathrm{M}}\, b \mid (\nu z)\,(\overline{c}\langle z\rangle.\,\overline{c}\langle a\rangle.\,[\![z := \lambda y.y]\!]^{\mathrm{M}})) & \to_{\pi}^{+} (c) \\
2: & (\nu z)\,([\![zz]\!]^{\mathrm{M}}\, a \mid [\![z := \lambda y.y]\!]^{\mathrm{M}}) & \triangleq \\
& (\nu z)\,((\nu c)\,([\![z]\!]^{\mathrm{M}}\, c \mid (\nu z_1)\,(\overline{c}\langle z_1\rangle.\,\overline{c}\langle a\rangle.\,[\![z_1 := z]\!]^{\mathrm{M}})) \mid [\![z := \lambda y.y]\!]^{\mathrm{M}}) & \triangleq \\
& (\nu z)\,((\nu c)\,(\overline{z}\langle c\rangle \mid (\nu z_1)\,(\overline{c}\langle z_1\rangle.\,\overline{c}\langle a\rangle.\,[\![z_1 := z]\!]^{\mathrm{M}})) \mid [\![z := \lambda y.y]\!]^{\mathrm{M}}) & \equiv \\
& (\nu z)\,((\nu c)\,(\overline{z}\langle c\rangle \mid z(w).\,[\![\lambda y.y]\!]^{\mathrm{M}}\, w \mid (\nu z_1)\,(\overline{c}\langle z_1\rangle.\,\overline{c}\langle a\rangle.\,[\![z_1 := z]\!]^{\mathrm{M}})) \mid [\![z := \lambda y.y]\!]^{\mathrm{M}}) & \to\ (z) \\
3: & (\nu z)\,((\nu c)\,([\![\lambda y.y]\!]^{\mathrm{M}}\, c \mid (\nu z_1)\,(\overline{c}\langle z_1\rangle.\,\overline{c}\langle a\rangle.\,[\![z_1 := z]\!]^{\mathrm{M}})) \mid [\![z := \lambda y.y]\!]^{\mathrm{M}}) & \triangleq \\
& (\nu z)\,((\nu c)\,(c(y).\,c(b).\,\overline{y}\langle b\rangle \mid (\nu z_1)\,(\overline{c}\langle z_1\rangle.\,\overline{c}\langle a\rangle.\,[\![z_1 := z]\!]^{\mathrm{M}})) \mid [\![z := \lambda y.y]\!]^{\mathrm{M}}) & \to_{\pi}^{+} (c) \\
4: & (\nu z)\,((\nu z_1)\,(\overline{z_1}\langle a\rangle \mid [\![z_1 := z]\!]^{\mathrm{M}}) \mid [\![z := \lambda y.y]\!]^{\mathrm{M}}) & \equiv \\
& (\nu z)\,((\nu z_1)\,(\overline{z_1}\langle a\rangle \mid z_1(w).\,\overline{z}\langle w\rangle \mid [\![z_1 := z]\!]^{\mathrm{M}}) \mid [\![z := \lambda y.y]\!]^{\mathrm{M}}) & \to\ (z_1) \\
5: & (\nu z)\,((\nu z_1)\,(\overline{z}\langle a\rangle \mid [\![z_1 := z]\!]^{\mathrm{M}}) \mid [\![z := \lambda y.y]\!]^{\mathrm{M}}) & \equiv \\
& (\nu z)\,((\nu z_1)\,(\overline{z}\langle a\rangle \mid [\![z_1 := z]\!]^{\mathrm{M}}) \mid z(w).[\![\lambda y.y]\!]^{\mathrm{M}}\, a \mid [\![z := \lambda y.y]\!]^{\mathrm{M}}) & \to\ (z) \\
6: & (\nu z)\,((\nu z_1)\,([\![\lambda y.y]\!]^{\mathrm{M}}\, a \mid [\![z_1 := z]\!]^{\mathrm{M}}) \mid [\![z := \lambda y.y]\!]^{\mathrm{M}}) & \equiv \\
& [\![\lambda y.y]\!]^{\mathrm{M}}\, a \mid (\nu z)\,((\nu z_1)\,([\![z_1 := z]\!]^{\mathrm{M}}) \mid [\![z := \lambda y.y]\!]^{\mathrm{M}}) & \sim_{\mathrm{C}} [\![\lambda y.y]\!]^{\mathrm{M}}\, a
\end{array}
$$

Figure 1: Running Milner's encoding of $(\lambda x.xx)(\lambda y.y)$.

So is there then a relation between Milner's encoding and explicit substitution? By close inspection it becomes clear that the reduction in Figure 1 actually simulates:

$$
\begin{array}{lllll}
(\lambda x.xx)(\lambda y.y) & \to_{\mathbf{x}\mathrm{L}} & (zz)\,\langle z := \lambda y.y\rangle & \to_{\mathbf{x}\mathrm{L}} & \\
 & & ((\lambda y.y)z)\,\langle z := \lambda y.y\rangle & \to_{\mathbf{x}\mathrm{L}} & \\
 & & z_1\,\langle z_1 := z\rangle\,\langle z := \lambda y.y\rangle & \to_{\mathbf{x}\mathrm{L}} & \\
 & & z\,\langle z_1 := z\rangle\,\langle z := \lambda y.y\rangle & \to_{\mathbf{x}\mathrm{L}} & (\lambda y.y)\,\langle z_1 := z\rangle\,\langle z := \lambda y.y\rangle
\end{array}
$$

(where the numbered lines in Figure 1 correspond to the respective terms in this reduction). Notice that this contains some unnecessary α-conversions, and that the congruence rules take care of part of the propagation of the substitution (in the last step before line 3); moreover, this is not a reduction in $\lambda\mathbf{x}$: the substitution $\langle x := \lambda y.y\rangle$ does not fully propagate in the second step, but only does so for the head-variable, and stays also on the outside, which gets only used when the innermost reduction has taken place.

We can now show that we can generalise the above observation, and show a (more direct) simulation result for Milner's encoding but now for explicit lazy reduction; first we need to extend that encoding to have it deal with explicit substitution as well.

Definition 17. We extend the interpretation of Definition 13 to $\lambda\mathbf{x}$ (and $\lambda\mathbf{x}\mathrm{L}$ and $\lambda\mathbf{x}\mathrm{S}$, for that matter) by adding the case:

$$[\![M\,\langle x := N\rangle]\!]^{\mathrm{M}}\, a \quad \triangleq \quad (\nu x)\,([\![M]\!]^{\mathrm{M}}\, a \mid [\![x := N]\!]^{\mathrm{M}})$$

Formulated using $\to_{\mathbf{x}\mathrm{L}}$, we can now show the following result for Milner's interpretation in the λ-calculus with explicit substitution:

Theorem 18 ($[\![\cdot]\!]^{\mathbf{M}}\cdot$ preserves $\to_{\mathbf{x}\mathrm{L}}$). *If $M \to_{\mathbf{x}\mathrm{L}}^{*} N$, then $[\![M]\!]^{\mathrm{M}}\, a \to_{\pi}^{+} [\![N]\!]^{\mathrm{M}}\, a$.*

PROOF. By induction on the definition of explicit lazy reduction; we only show the basic cases.

$$
\begin{array}{lll}
(\lambda y.M)N \to_{\mathbf{x}\mathrm{L}} M\,\langle y := N\rangle: \quad [\![(\lambda y.M)N]\!]^{\mathrm{M}}\, a & \triangleq & \\
\quad (\nu c)\,(c(y).\,c(b).\,[\![M]\!]^{\mathrm{M}}\, b \mid (\nu z)\,(\overline{c}\langle z\rangle.\,\overline{c}\langle a\rangle.\,[\![z := N]\!]^{\mathrm{M}})) & \to_{\pi}^{+} & (c) \\
\quad (\nu z)\,([\![M[z/y]]\!]^{\mathrm{M}}\, a \mid [\![z := N]\!]^{\mathrm{M}}) & =_{\alpha} & \\
\quad (\nu y)\,([\![M]\!]^{\mathrm{M}}\, a \mid [\![y := N]\!]^{\mathrm{M}}) & \triangleq & [\![M\,\langle y := N\rangle]\!]^{\mathrm{M}}\, a
\end{array}
$$

$(xM_1\cdots M_n\,\overrightarrow{\langle y:=L\rangle})\,\langle x:=N\rangle \to_{\mathbf{xL}} (NM_1\cdots M_n\,\overrightarrow{\langle y:=L\rangle})\,\langle x:=N\rangle$:

$$
\begin{array}{lll}
[\![(xM_1\cdots M_n\,\overrightarrow{\langle y:=L\rangle})\,\langle x:=N\rangle]\!]^{\mathrm{M}}\,a & \triangleq & \\
(\nu x)\,([\![xM_1\cdots M_n\,\overrightarrow{\langle y:=L\rangle}]\!]^{\mathrm{M}}\,a \mid [\![x:=N]\!]^{\mathrm{M}}) & \triangleq & \\
(\nu x)\,((\nu c)\,([\![xM_1\cdots M_{n-1}]\!]^{\mathrm{M}}\,c \mid (\nu z)\,(\overline{c}\langle z\rangle.\,\overline{c}\langle a\rangle.\,[\![z:=M_n]\!]^{\mathrm{M}})) \mid \overrightarrow{[\![y:=L]\!]^{\mathrm{M}}} \mid [\![x:=N]\!]^{\mathrm{M}}) & \triangleq, \equiv & \\
(\nu x\vec{y}cc_1\cdots c_{n-1})\,(\overline{x}\langle c_{n-1}\rangle \mid (\nu z_{n-1})\,(\overline{c_{n-1}}\langle z_{n-1}\rangle.\,\overline{c_{n-1}}\langle c\rangle.\,[\![z_{n-1}:=M_1]\!]^{\mathrm{M}}) \mid \cdots \mid & & \\
\qquad (\nu z)\,(\overline{c}\langle z\rangle.\,\overline{c}\langle a\rangle.\,[\![z:=M_n]\!]^{\mathrm{M}}) \mid \overrightarrow{[\![y:=L]\!]^{\mathrm{M}}} \mid [\![x:=N]\!]^{\mathrm{M}}) & \equiv & \\
(\nu x\vec{y}cc_1\cdots c_{n-1})\,(\overline{x}\langle c_{n-1}\rangle \mid (\nu z_{n-1})\,(\overline{c_{n-1}}\langle z_{n-1}\rangle.\,\overline{c_{n-1}}\langle c\rangle.\,[\![z_{n-1}:=M_1]\!]^{\mathrm{M}}) \mid \cdots \mid & & \\
\qquad (\nu z)\,(\overline{c}\langle z\rangle.\,\overline{c}\langle a\rangle.\,[\![z:=M_n]\!]^{\mathrm{M}}) \mid \overrightarrow{[\![y:=L]\!]^{\mathrm{M}}} \mid x(w).\,[\![N]\!]^{\mathrm{M}}\,w \mid [\![x:=N]\!]^{\mathrm{M}}) & \to_{\pi} \quad (x) & \\
(\nu x\vec{y}cc_1\cdots c_{n-1})\,([\![N]\!]^{\mathrm{M}}\,c_{n-1} \mid (\nu z_{n-1})\,(\overline{c_{n-1}}\langle z_{n-1}\rangle.\,\overline{c_{n-1}}\langle c\rangle.\,[\![z_{n-1}:=M_1]\!]^{\mathrm{M}}) \mid \cdots \mid & & \\
\qquad (\nu z)\,(\overline{c}\langle z\rangle.\,\overline{c}\langle a\rangle.\,[\![z:=M_n]\!]^{\mathrm{M}}) \mid \overrightarrow{[\![y:=L]\!]^{\mathrm{M}}} \mid [\![x:=N]\!]^{\mathrm{M}}) & \equiv, \triangleq & \\
[\![(NM_1\cdots M_n\,\overrightarrow{\langle y:=L\rangle})\,\langle x:=N\rangle]\!]^{\mathrm{M}}\,a & &
\end{array}
$$

$M \to_{\mathbf{xL}} M' \Rightarrow MN \to_{\mathbf{xL}} M'N$:

$$
\begin{array}{lll}
[\![MN]\!]^{\mathrm{M}}\,a & \triangleq & \\
(\nu c)\,([\![M]\!]^{\mathrm{M}}\,c \mid (\nu z)\,(\overline{c}\langle z\rangle.\,\overline{c}\langle a\rangle.\,[\![z:=N]\!]^{\mathrm{M}})) & \to_{\pi}^{+} \quad (IH) & \\
(\nu c)\,([\![M']\!]^{\mathrm{M}}\,c \mid (\nu z)\,(\overline{c}\langle z\rangle.\,\overline{c}\langle a\rangle.\,[\![z:=N]\!]^{\mathrm{M}})) & \triangleq \quad [\![M'N]\!]^{\mathrm{M}}\,a &
\end{array}
$$

$M \to_{\mathbf{xL}} M' \Rightarrow M\,\langle x:=N\rangle \to_{\mathbf{xL}} M'\,\langle x:=N\rangle$:

$$
\begin{array}{lll}
[\![M\,\langle x:=N\rangle]\!]^{\mathrm{M}}\,a & \triangleq & \\
(\nu x)\,([\![M]\!]^{\mathrm{M}}\,a \mid [\![x:=N]\!]^{\mathrm{M}}) \;\to_{\pi}^{+}\; (IH) \;\; (\nu x)\,([\![M']\!]^{\mathrm{M}}\,a \mid [\![x:=N]\!]^{\mathrm{M}}) & \triangleq & [\![M'\,\langle x:=N\rangle]\!]^{\mathrm{M}}\,a
\end{array}
$$

□

Notice that, in particular, we do not need the λ-terms involved to be closed, but of course can show:

Corollary 19. *If M is closed, and $M \to_{\mathbf{xL}}^{*} (\lambda y.N)\overrightarrow{\langle x:=L\rangle}$, then $[\![M]\!]^{\mathrm{M}}\,a \to_{\pi}^{+} [\![\lambda y.N]\!]^{\mathrm{M}}\,a \mid \overrightarrow{[\![y:=L]\!]^{\mathrm{M}}}$.*

which is Milner's main result, but stated using explicit substitution, as we think it should have been.

We can of course also restate these results using normal lazy reduction:

Theorem 20. 1. *If $M \to_{\mathrm{L}}^{*} N$, then there exists M' such that $[\![M]\!]^{\mathrm{M}}\,a \to_{\pi}^{+} [\![M']\!]^{\mathrm{M}}\,a$, and $M' \to_{:=} N$.*

2. *If M is closed and lazy normalising, then there exists $N, \vec{x}, \vec{L}$ such that $M \to_{\mathrm{L}} (\lambda y.N)\overrightarrow{[L/x]}$, and $[\![M]\!]^{\mathrm{M}}\,a \to_{\pi}^{+} [\![\lambda y.N]\!]^{\mathrm{M}}\,a \mid \overrightarrow{[\![y:=L]\!]^{\mathrm{M}}}$.*

PROOF. By Theorem 18 and Proposition 12. □

After Milner's original encoding, many variants followed, with perhaps the most expressive being the uniform encoding $[\![\cdot]\!]^{\mathrm{U}}\cdot$, as defined in [26]. That encoding is, in particular, engineered to work in Call-By-Name, Call-By-Value, and Call-By-Need; here we focus on the CBN component, and adapt the definition to our notation.

Definition 21 (The uniform encoding [26]). The CBN-variant of the uniform encoding of the λ-calculus into the (synchronous) π-calculus with pairing is defined as follows:

$$
\begin{array}{rcl}
[\![x]\!]^{\mathrm{U}}\,p & \triangleq & \overline{x}\langle p\rangle \\
[\![\lambda x.M]\!]^{\mathrm{U}}\,p & \triangleq & (\nu v)\,(\overline{p}\langle v\rangle.\,!\,v(x,q).\,[\![M]\!]^{\mathrm{U}}\,q) \\
[\![MN]\!]^{\mathrm{U}}\,p & \triangleq & (\nu q)\,([\![M]\!]^{\mathrm{U}}\,q \mid q(v).\,(\nu x)\,(\overline{v}\langle x,p\rangle.\,!\,x(w).\,[\![N]\!]^{\mathrm{U}}\,w))
\end{array}
$$

This encoding contains, at least for CBN, some superfluous replication in the case that deals with abstraction, but this is needed to successfully encode the other two reduction strategies. Although this is an *output*-based encoding, in the sense that the (private) channel q in the encoding of MN is used as an *output* for the encoding of M, underneath the encoding is essentially Milner's.

Example 22. In general, the encoding of a redex runs as follows:

$$\begin{array}{lll}
[\![(\lambda y.M)N]\!]^{\mathrm{U}} p & \triangleq & \\
(\nu q)\,([\![\lambda y.M]\!]^{\mathrm{U}} q \mid q(v).\,(\nu x)\,(\overline{v}\langle x,p\rangle.\,!\,x(w).\,[\![N]\!]^{\mathrm{U}} w)) & \triangleq & \\
(\nu q)\,((\nu v)\,(\overline{q}\langle v\rangle.\,!\,v(y,r).\,[\![M]\!]^{\mathrm{U}} r) \mid q(v).\,(\nu x)\,(\overline{v}\langle x,p\rangle.\,!\,x(w).\,[\![N]\!]^{\mathrm{U}} w)) & \to_\pi & (q)\\
(\nu v)\,(!\,v(y,r).\,[\![M]\!]^{\mathrm{U}} r \mid (\nu x)\,(\overline{v}\langle x,p\rangle.\,!\,x(w).\,[\![N]\!]^{\mathrm{U}} w)) & \to_\pi & (v)\\
(\nu x v)\,(!\,v(y,r).\,[\![M]\!]^{\mathrm{U}} r \mid [\![M[x/y]]\!]^{\mathrm{U}} p \mid !\,x(w).\,[\![N]\!]^{\mathrm{U}} w) & =_\alpha & \\
(\nu y v)\,(!\,v(y,r).\,[\![M]\!]^{\mathrm{U}} r \mid [\![M]\!]^{\mathrm{U}} p \mid !\,y(w).\,[\![N]\!]^{\mathrm{U}} w) & \equiv & \\
(\nu y)\,((\nu v)\,(!\,v(y,r).\,[\![M]\!]^{\mathrm{U}} r) \mid [\![M]\!]^{\mathrm{U}} p \mid !\,y(w).\,[\![N]\!]^{\mathrm{U}} w) & \sim_\pi & \\
(\nu y)\,([\![M]\!]^{\mathrm{U}} p \mid !\,y(w).\,[\![N]\!]^{\mathrm{U}} w) & &
\end{array}$$

In particular:

$$\begin{array}{lll}
[\![(\lambda y.y)(\lambda z.z)]\!]^{\mathrm{U}} p & \to_\pi\ (q,v), \sim_\pi & \\
(\nu y)\,([\![y]\!]^{\mathrm{U}} p \mid !\,y(w).\,[\![\lambda z.z]\!]^{\mathrm{U}} w) & \triangleq, \equiv & \\
(\nu y)\,(\overline{y}\langle p\rangle \mid y(w).\,[\![\lambda z.z]\!]^{\mathrm{U}} w \mid !\,y(w).\,[\![\lambda z.z]\!]^{\mathrm{U}} w) & \to_\pi\ (y) & \\
(\nu y)\,([\![\lambda z.z]\!]^{\mathrm{U}} p \mid !\,y(w).\,[\![\lambda z.z]\!]^{\mathrm{U}} w) & \equiv & \\
[\![\lambda z.z]\!]^{\mathrm{U}} p \mid (\nu y)\,(!\,y(w).\,[\![\lambda z.z]\!]^{\mathrm{U}} w) & \sim_\pi & [\![\lambda z.z]\!]^{\mathrm{U}} p
\end{array}$$

Notice that this reduction, after the synchronisation over q, essentially runs like Milner's encoding, but uses pairing as in Section 4.

This example allows us to extend the uniform CBN-encoding to $\lambda\mathbf{x}$ by adding:

$$[\![M\langle x := N\rangle]\!]^{\mathrm{U}} p \quad \triangleq \quad (\nu x)\,([\![M]\!]^{\mathrm{U}} p \mid [\![x := N]\!]^{\mathrm{U}})$$

where, inspired by Milner's encoding, we write $[\![x := N]\!]^{\mathrm{U}}$ for $!\,x(w).\,[\![N]\!]^{\mathrm{U}} w$.

Notice that, as before, the reductions inside an abstraction, those in the right-hand side of an application, as well as those inside the term that gets substituted, cannot be simulated, and that, therefore, this encoding models (part of) lazy reduction. Moreover, only the head variable can get replaced; in fact, it is easy to show that also this encoding respects explicit lazy reduction:

Theorem 23. *If $M \to_{\mathbf{x}\mathrm{L}}^{*} N$, then $[\![M]\!]^{\mathrm{U}} p \to_\pi^{+} [\![N]\!]^{\mathrm{U}} p$.*

PROOF. By induction on the definition of explicit lazy reduction; we only show the basic cases.

$(\lambda x.P)Q \to_{\mathbf{x}\mathrm{L}} P\langle x := Q\rangle$: By Example 22.

$(zM_1\cdots M_n\,\overrightarrow{\langle y := L\rangle})\langle z := N\rangle \to_{\mathbf{x}\mathrm{L}} (NM_1\cdots M_n\,\overrightarrow{\langle y := L\rangle})\langle z := N\rangle$:

$$\begin{array}{lll}
[\![(zM_1\cdots M_n\,\overrightarrow{\langle y := L\rangle})\langle z := N\rangle]\!]^{\mathrm{U}} p & \triangleq & \\
(\nu z\vec{y})\,((\nu q)\,([\![zM_1\cdots M_{n-1}]\!]^{\mathrm{U}} q \mid q(v).\,(\nu x)\,(\overline{v}\langle x,p\rangle.\,[\![x := M_n]\!]^{\mathrm{U}})) \mid & & \\
\qquad\qquad \overrightarrow{[\![y := L]\!]^{\mathrm{U}}} \mid [\![z := N]\!]^{\mathrm{U}}) & \triangleq, \equiv & \\
(\nu z\vec{y}qq_1q_2\cdots q_n)\,(\overline{z}\langle q_n\rangle \mid q_n(v_n).\,(\nu x_n)\,(\overline{v_n}\langle x_n, q_{n-1}\rangle.\,[\![x_n := M_1]\!]^{\mathrm{U}})) \mid \cdots \mid & & \\
\qquad q(v).\,(\nu x)\,(\overline{v}\langle x,p\rangle.\,[\![x := M_n]\!]^{\mathrm{U}})) \mid \overrightarrow{[\![y := L]\!]^{\mathrm{U}}} \mid z(w).\,[\![N]\!]^{\mathrm{U}} w \mid [\![z := N]\!]^{\mathrm{U}}) & \to_\pi\ (z) & \\
(\nu z\vec{y}qq_1q_2\cdots q_n)\,([\![N]\!]^{\mathrm{U}} q_n \mid q_n(v_n).\,(\nu x_n)\,(\overline{v_n}\langle x_n, q_{n-1}\rangle.\,[\![x_n := M_1]\!]^{\mathrm{U}})) \mid \cdots \mid & & \\
\qquad q(v).\,(\nu x)\,(\overline{v}\langle x,p\rangle.\,[\![x := M_n]\!]^{\mathrm{U}})) \mid \overrightarrow{[\![y := L]\!]^{\mathrm{U}}} \mid [\![z := N]\!]^{\mathrm{U}}) & \equiv, \triangleq & \\
[\![(NM_1\cdots M_n\,\overrightarrow{\langle y := L\rangle})\langle z := N\rangle]\!]^{\mathrm{U}} p & &
\end{array}$$

$M \to_{\mathbf{x}\mathrm{L}} M' \Rightarrow MN \to_{\mathbf{x}\mathrm{L}} M'N$, $M\langle z := N\rangle \to_{\mathbf{x}\mathrm{L}} M'\langle z := N\rangle$: By induction. □

Notice that, in particular, as in Theorem 18, we do not need the λ-terms involved in the reduction to be closed, and that we model step-by-step explicit lazy reduction.

The following then becomes immediate:

Theorem 24. *If $M \to_{\mathrm{L}} N$, then there exists N' such that $[\![M]\!]^{\mathrm{U}} p \to_{\pi}^{+} [\![N']\!]^{\mathrm{U}} p$, and $N' \to_{\mathbf{x}\mathrm{L}}^{*} N$.*

PROOF. By Theorem 23 and Proposition 12. □

4. A logical, *output*-based encoding of λ-terms

In this section, we will show that it is possible to deviate from Milner's original approach to encoding, and actually make a gain in the process. Inspired by the relation between natural deduction and the sequent calculus [13], interpreting terms under *output* rather than under *input*, and using the π-calculus with pairing, we can define a different encoding of the λ-calculus into the π-calculus. Although the main objective of our encoding is to show the preservation of type assignment, and pairing is used in order to be able to effectively represent arrow types, we also achieve a more expressive encoding that preserves not just lazy reduction, but also the larger notion of spine reduction.

Our encoding follows from – but is an improvement of – the concatenation of the encoding of the λ-calculus into $\mathcal{X}$ (which established a link between natural deduction and the sequent calculus) as defined in [6], and the interpretation of $\mathcal{X}$ into the π-calculus as defined in [4]. The idea behind our encoding originates from the observation that, in the λ-calculus, all *input* is named, but *output* is anonymous. *Input* (i.e., a *variable*) is named to serve as a destination for the substitution; *output* need not be named, since all terms have only one result (represented by the term itself), which is used *in sito*[12]. Translating into the (multi-*output*) π-calculus, this locality property no longer holds; we need to specify the destination of a term, by naming its *output*: this is what the encoding does.

We explicitly convert '*an output sent on a is to be received as input on b*' via '$a(w).\overline{b}\langle w\rangle$' (called a *forwarder* in [16]), which for convenience is denoted by $a \twoheadrightarrow b$.

Definition 25 (*Output*-based interpretation of the λ-calculus in π). The mapping $[\![\cdot]\!]\cdot$ is defined by:

$$\begin{array}{rcll}
[\![x]\!]a & \triangleq & x(z).z(w).\overline{a}\langle w\rangle & x \neq a \\
[\![\lambda x.M]\!]a & \triangleq & (\nu xb)\,([\![M]\!]b \mid \overline{a}\langle x,b\rangle) & b \textit{ fresh} \\
[\![MN]\!]a & \triangleq & (\nu c)\,([\![M]\!]c \mid c(b,d).(!\,(\nu w)\,(\overline{b}\langle w\rangle.[\![N]\!]w) \mid d \twoheadrightarrow a)) & b,c,d \textit{ fresh}
\end{array}$$

As for Milner's encoding, we use the abbreviation

$$[\![x := N]\!] \quad \triangleq \quad !\,(\nu w)\,(\overline{x}\langle w\rangle.[\![N]\!]w)$$

Notice that the encoding is not trivial, since

$$\begin{array}{rcl}
[\![\lambda yz.y]\!]a & = & (\nu yb)\,((\nu zb_1)\,(y(z).z(w).\overline{b_1}\langle w\rangle \mid \overline{b_1}\langle z,b_1\rangle) \mid \overline{a}\langle y,b\rangle) \\
[\![\lambda x.x]\!]a & = & (\nu xb)\,(x(z).z(w).\overline{b}\langle w\rangle \mid \overline{a}\langle x,b\rangle)
\end{array}$$

processes that differ under $\sim_{\mathrm{C}}$.

In particular, notice that the operand N in the application MN is interpreted by a guarded replication: N itself gets interpreted under the (*output*) name w, which gets sent out first over the channel b, which is the *input* channel of M, if any. So N can only run after synchronisation over b has taken place, *i.e.* if an *input* for (the channel name that will replace) b exists. Running the encoding will eventually simulate the substitution $x\,\langle x := N\rangle$ via

$$\begin{array}{rcll}
[\![x]\!]a \mid (\nu v)\,(\overline{x}\langle v\rangle.[\![N]\!]v) & \triangleq & (\nu v)\,(x(z).z(w).\overline{a}\langle w\rangle \mid \overline{x}\langle v\rangle.[\![N]\!]v) & \to_{\pi} \quad (x) \\
 & & (\nu v)\,(v(w).\overline{a}\langle w\rangle \mid [\![N]\!]v) & \sim_{\mathrm{C}} \quad [\![N]\!]a
\end{array}$$

We see the synchronisation over x as the effectuation of the substitution. Also:

[12] In terms of *continuations*, the continuation of a term is not mentioned, since it is the current.

- We see a variable x as an *input* channel, over which we receive the *output* name of the process that needs to be substituted for x; we use this name to create an *input* channel to receive the *output* of N, which we then send out of the *output* name of x, being a;
- For an abstraction $\lambda x.M$, we give the name b to the *output* of M; that M has *input* x and *output* b gets sent out over a, which is the name of $\lambda x.M$, so that a process that wants to call on this functionality, knows which channel to send the *input* to, and on which channel to pick up the result[13];
- For an application MN, the *output* of M, transmitted over c, is received as a pair $\langle b, d\rangle$ of *input*-*output* names in the right-hand side; the received *input* b name is used as send the fresh *output* name for N, enabling the simulation of substitution, and the received *output* name d gets redirected to the *output* of the application a.

Notice that only one replication is used, on the argument in an application; this corresponds, as above, to the implementation of the (distributive) substitution on λ-terms. Also, every $\llbracket N \rrbracket a$ is a process that *output*s on a non-hidden name a (albeit perhaps not actively, as in the third case, where it will not be activated until *input* is received on the channel c, in which case it is used to *output* the data received in on the channel d that is passed as a parameter), and that this *output* is *unique*, in the sense that a is the only *output* channel, is only used once, and for *output* only. This implies that $(\nu a)\,\llbracket M \rrbracket a \sim_{\mathrm{C}} \mathbf{0}$, for all M; these are not bi-similar, of course.

The structure of the encoding of application corresponds, in fact, to how Gentzen encodes *modus ponens* in the sequent calculus [13]: see [6], Theorem 4.8, and the proof of Theorem 48 below.

Example 26.

$$
\begin{array}{ll}
\llbracket (\lambda p.p)(\lambda x.x) \rrbracket a & \triangleq \\
(\nu c)\,((\nu p b_1)\,(p(z).z(w).\overline{b_1}\langle w\rangle \mid \overline{c}\langle p, b_1\rangle) \mid c(b,d).(\llbracket b := \lambda x.x \rrbracket \mid d \twoheadrightarrow a)) & \rightarrow_\pi\ (c) \\
(\nu p b_1)\,(p(z).z(w).\overline{b_1}\langle w\rangle \mid \llbracket p := \lambda x.x \rrbracket \mid b_1 \twoheadrightarrow a) & \equiv \\
(\nu p b_1)\,(p(z_1).z_1(w).\overline{b_1}\langle w\rangle \mid (\nu y)\,(\overline{p}\langle y\rangle.\llbracket \lambda x.x \rrbracket y) \mid \llbracket p := \lambda x.x \rrbracket \mid b_1 \twoheadrightarrow a) & \rightarrow_\pi\ (p) \\
(\nu b_1 y)\,(y(w).\overline{b_1}\langle w\rangle \mid \llbracket \lambda x.x \rrbracket y \mid (\nu p)\,(\llbracket p := \lambda x.x \rrbracket) \mid b_1 \twoheadrightarrow a) & \triangleq \\
(\nu b_1 y)\,(y(w).\overline{b_1}\langle w\rangle \mid (\nu x b)\,(\llbracket x \rrbracket b \mid \overline{y}\langle x, b\rangle) \mid (\nu p)\,(\llbracket p := \lambda x.x \rrbracket) \mid b_1 \twoheadrightarrow a) & \rightarrow_\pi^+\ (y, b_1) \\
(\nu x b)\,(\llbracket x \rrbracket b \mid \overline{a}\langle x, b\rangle) \mid (\nu p)\,(\llbracket p := \lambda x.x \rrbracket) & \triangleq \\
\llbracket \lambda x.x \rrbracket a \mid (\nu p)\,(\llbracket p := \lambda x.x \rrbracket) & \sim_{\mathrm{C}}\ \llbracket \lambda x.x \rrbracket a
\end{array}
$$

That we use asynchronous synchronisation in our encoding of abstraction is not only convenient – since it allows us to express not just lazy reduction, but also spine reduction as well, as we will show in the next section – it is also necessary:

Example 27. Assume that $\llbracket \lambda x.M \rrbracket a = (\nu x b)\,(\overline{a}\langle x, b\rangle.\llbracket M \rrbracket b)$, then $\llbracket (\lambda y.y)(\lambda x.x) \rrbracket a$ would run as follows:

$$
\begin{array}{lll}
\llbracket (\lambda y.y)(\lambda x.x) \rrbracket a & \triangleq & \\
(\nu c)\,(\llbracket \lambda y.y \rrbracket c \mid c(b,d).(\llbracket b := (\lambda x.x) \rrbracket \mid d \twoheadrightarrow a)) & \triangleq & \\
(\nu c)\,((\nu y b_1)\,(\overline{c}\langle y, b_1\rangle.\llbracket y \rrbracket b_1) \mid c(b,d).(\llbracket b := (\lambda x.x) \rrbracket \mid d \twoheadrightarrow a)) & \rightarrow_\pi & (c) \\
(\nu y b_1)\,(\llbracket y \rrbracket b_1 \mid \llbracket y := \lambda x.x \rrbracket \mid b_1 \twoheadrightarrow a) & \equiv, \triangleq & \\
(\nu y b_1)\,(y(z).z(w).\overline{b_1}\langle w\rangle \mid (\nu v)\,(\overline{y}\langle v\rangle.(\nu x b)\,(\overline{v}\langle x, b\rangle.\llbracket x \rrbracket b)) \mid \llbracket y := \lambda x.x \rrbracket \mid b_1 \twoheadrightarrow a) & \rightarrow_\pi & (y, v, b_1) \\
(\nu y x b)\,(\overline{a}\langle x, b\rangle \mid \llbracket x \rrbracket b \mid \llbracket y := \lambda x.x \rrbracket) & \equiv & \\
(\nu x b)\,(\overline{a}\langle x, b\rangle \mid \llbracket x \rrbracket b) \mid (\nu y)\,(\llbracket y := \lambda x.x \rrbracket) & \sim_{\mathrm{C}} & \\
(\nu x b)\,(\overline{a}\langle x, b\rangle \mid \llbracket x \rrbracket b) & &
\end{array}
$$

Notice that then the last term would not correspond to $\llbracket \lambda x.x \rrbracket a$; this clearly shows that we generate an asynchronous *output* in the interpretation for the result of the reduction. To obtain an encoding that respects reduction, we therefore had to define it using asynchronous *output* for the abstraction.

[13] This view of computation is exactly that of the calculus $\mathcal{X}$.

For the encoding presented here, the encoding of a redex runs as follows:

Example 28.
$$\begin{array}{ll} \llbracket(\lambda y.P)Q\rrbracket a & \triangleq \\ (\nu c)\,((\nu y b_1)\,(\llbracket P\rrbracket b_1 \mid \overline{c}\langle y, b_1\rangle) \mid c(b,d).\,(\llbracket b := Q\rrbracket \mid d \twoheadrightarrow a)) & \rightarrow_\pi \\ (\nu y b_1)\,(\llbracket P\rrbracket b_1 \mid \llbracket y := Q\rrbracket \mid b_1 \twoheadrightarrow a) & \end{array}$$

The encoding of the redex $(\lambda y.P)Q$ will yield a process containing a synchronisation that receives on the *input* channel called y in the interpretation of P, and the interpretation of Q being *output* on y (see Theorem 32). Since $\llbracket P\rrbracket b_1$ is sending its *output* on b_1, which gets redirected to a, we will see this as similar to $(\nu y)\,(\llbracket P\rrbracket a \mid \llbracket y := Q\rrbracket)$ (see Lemma 30).

A first version of our encoding was presented in [4], where it was obtained by the concatenation of the encoding $\llbracket\cdot\rrbracket^{\lambda}$ of the λ-calculus into the sequent calculus $\mathcal{X}$ as defined in [6], and the encoding $[\![\cdot]\!]$ of $\mathcal{X}$ into the π-calculus, as presented in that paper. This resulted in an interpretation $\llbracket\cdot\rrbracket\cdot : \Lambda\rightarrow\pi$ of the λ-calculus in π via $\mathcal{X}$ as defined by: $\llbracket M\rrbracket\alpha = [\![\llbracket M\rrbracket^{\lambda}_{\alpha}]\!]$, i.e.:

$$\begin{array}{rcll} \llbracket x\rrbracket\alpha & \triangleq & x(w).\overline{\alpha}\langle w\rangle & \\ \llbracket\lambda x.M\rrbracket\alpha & \triangleq & (\nu x\beta)\,(!\,\llbracket M\rrbracket\beta \mid \overline{\alpha}\langle x,\beta\rangle), & \beta \textit{ fresh} \\ \llbracket MN\rrbracket\alpha & \triangleq & (\nu\delta c)\,(!\,\llbracket M\rrbracket\delta \mid !\,\delta\twoheadrightarrow c \mid !\,c(v,d).\,(\nu\beta)\,!\,\llbracket N\rrbracket\beta \mid \beta\twoheadrightarrow v \mid d\twoheadrightarrow\alpha), & \gamma,\beta,x,y \textit{ fresh} \end{array}$$

Using, for example, the insight that $\llbracket M\rrbracket\delta \mid \delta\twoheadrightarrow c$ is in fact computationally the same as $\llbracket M\rrbracket c$, and that replication is only needed for substitution, we simplified this encoding to the one we presented in [7]:

$$\begin{array}{rcll} \llbracket x\rrbracket a & \triangleq & x(w).\overline{a}\langle w\rangle & x \neq a \\ \llbracket\lambda x.M\rrbracket a & \triangleq & (\nu xb)\,(\llbracket M\rrbracket b \mid \overline{a}\langle x,b\rangle) & b \textit{ fresh} \\ \llbracket MN\rrbracket a & \triangleq & (\nu c)\,(\llbracket M\rrbracket c \mid c(b,d).\,(!\,\llbracket N\rrbracket b \mid d\twoheadrightarrow a)) & b,c,d \textit{ fresh} \end{array}$$

For this encoding, we can show all the major properties that we show for the encoding of Definition 25, but for Theorem 40; in particular, we can show that, if $M \rightarrow^*_{\mathbf{xS}} N$, then $\llbracket M\rrbracket a \sim_{\mathrm{C}} \llbracket N\rrbracket a$, so these processes have the same observable behaviour; this implies that any non-termination takes place inside a process that has no visible *output*, i.e. is *unobservable*[14]. Using the encoding of Definition 25, we can show preservation of termination as well.

We could have defined our encoding directly in the standard synchronous π-calculus:

$$\begin{array}{rcl} \llbracket x\rrbracket' a & \triangleq & x(z).\,!\,z(w).\overline{a}\langle w\rangle \\ \llbracket\lambda x.M\rrbracket' a & \triangleq & (\nu xb)\,(\llbracket M\rrbracket' b \mid \overline{a}\langle x\rangle \mid \overline{a}\langle b\rangle) \\ \llbracket MN\rrbracket' a & \triangleq & (\nu c)\,(\llbracket M\rrbracket' c \mid c(b).\,c(d).\,(!\,(\nu w)\,(\overline{b}\langle w\rangle.\,\llbracket N\rrbracket' w) \mid !\,d\twoheadrightarrow a)) \end{array}$$

without losing the main reduction results for our encoding that we will show below, but this has additional replication, is less suited for type assignment (see Section 7), and does not satisfy some of the properties we consider here.

To show the need for replication in this variant also on the interpretation of variables in $\llbracket\cdot\rrbracket'\cdot$ and the forwarder, consider the reduction in Figure 2. This reduction shows that the parallel composition of the outputs on a in $(\nu xb)\,(\llbracket M\rrbracket' b \mid \overline{a}\langle x\rangle \mid \overline{a}\langle b\rangle)$ is necessary, similar to Example 27, as is the use of replication in the interpretation of the variable. Since *output* is generated twice over v, the receiving side has to run twice, so we need replication inside the interpretation of a variable; since we send twice on b_1, also the forwarder needs to be replicated. However, notice that these can now execute in arbitrary order, resulting in the wrong parameter exchange; so we can no longer guarantee operational completeness (see Theorem 39).

[14]This is a common fact in semantic interpretations: also the encoding of recursive programs into the λ-calculus does non respect termination; this uses a fixed-point construction to represent recursion, typically via the term $\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$, which already on its own does not terminate, leaving encoded functions with non-terminating, but unobservable parts.

$$
\begin{array}{ll}
\llbracket(\lambda y.y)(\lambda x.x)\rrbracket' a & \triangleq \\
(\nu c)\,(\llbracket\lambda y.y\rrbracket' c \mid c(b).c(d).(!\,(\nu w)\,(\overline{b}\langle w\rangle.\llbracket\lambda x.x\rrbracket' w) \mid !\,d{\twoheadrightarrow}a)) & \triangleq \\
(\nu c)\,((\nu y b_1)\,(\llbracket y\rrbracket' b_1 \mid \overline{c}\langle y\rangle \mid \overline{c}\langle b_1\rangle) \mid c(b).c(d).(!\,(\nu w)\,(\overline{b}\langle w\rangle.\llbracket\lambda x.x\rrbracket' w) \mid !\,d{\twoheadrightarrow}a)) & \to_\pi\ (c)\ (2\times) \\
(\nu y b_1)\,(\llbracket y\rrbracket' b_1 \mid !\,(\nu v)\,(\overline{y}\langle v\rangle.\llbracket\lambda x.x\rrbracket' v) \mid !\,b_1{\twoheadrightarrow}a) & \equiv, \triangleq \\
(\nu y b_1)\,(y(z).!\,z(w).\overline{b_1}\langle w\rangle \mid (\nu v)\,(\overline{y}\langle v\rangle.\llbracket\lambda x.x\rrbracket' v) \mid !\,(\nu v)\,(\overline{y}\langle v\rangle.\llbracket\lambda x.x\rrbracket' v) \mid !\,b_1{\twoheadrightarrow}a) & \to_\pi\ (y) \\
(\nu b_1 v)\,(!\,v(w).\overline{b_1}\langle w\rangle \mid (\nu x b)\,(\llbracket x\rrbracket' b \mid \overline{v}\langle x\rangle \mid \overline{v}\langle b\rangle) \mid !\,(\nu v)\,(\overline{y}\langle v\rangle.\llbracket\lambda x.x\rrbracket' v) \mid !\,b_1{\twoheadrightarrow}a) & \to_\pi\ (v)\ (2\times) \\
(\nu b_1 v b x)\,(!\,v(w).\overline{b_1}\langle w\rangle \mid \overline{b_1}\langle x\rangle \mid \overline{b_1}\langle b\rangle \mid \llbracket x\rrbracket' b \mid !\,(\nu v)\,(\overline{y}\langle v\rangle.\llbracket\lambda x.x\rrbracket' v) \mid !\,b_1{\twoheadrightarrow}a) & \to_\pi\ (b_1)\ (2\times) \\
(\nu b_1 v b x)\,(!\,v(w).\overline{b_1}\langle w\rangle \mid \overline{a}\langle x\rangle \mid \overline{a}\langle b\rangle \mid \llbracket x\rrbracket' b \mid !\,(\nu v)\,(\overline{y}\langle v\rangle.\llbracket\lambda x.x\rrbracket' v) \mid !\,b_1{\twoheadrightarrow}a) & \sim_\pi \\
(\nu x b)\,(\llbracket x\rrbracket' b \mid \overline{a}\langle x\rangle \mid \overline{a}\langle b\rangle) & \triangleq\ \ \llbracket\lambda x.x\rrbracket' a
\end{array}
$$

Figure 2: Running $\llbracket(\lambda y.y)(\lambda x.x)\rrbracket' a$

5. Preservation of spine reduction

We will now show that our encoding respects explicit spine reduction. By the very nature of that encoding this result is not exactly *" If $M \to_{\mathbf{x}\mathrm{S}} N$, then $\llbracket M\rrbracket p \to_\pi^+ \llbracket N\rrbracket p$ "* but instead gets formulated via a relation that also permits renaming of *output*. Renaming is defined and justified via the following lemma, which states that we can safely rename the (hidden) *output* of an encoded λ-term, and is needed below:

Lemma 29. $(\nu a)\,(c(b,d).(!\,\overline{b}\langle w\rangle.\llbracket M\rrbracket w \mid d{\twoheadrightarrow}a) \mid a{\twoheadrightarrow}e) \quad \sim_{\mathrm{C}} \quad c(b,d).(\nu a)\,(!\,\overline{b}\langle w\rangle.\llbracket M\rrbracket w \mid d{\twoheadrightarrow}a \mid a{\twoheadrightarrow}e)$

PROOF. Any context that interacts with the either the left or the right-hand side has to do so via $\overline{c}\langle b', d'\rangle$; in both cases this yields a process equivalent to $(\nu a)\,(!\,\overline{b'}\langle w\rangle.\llbracket M\rrbracket w \mid d'{\twoheadrightarrow}a \mid a{\twoheadrightarrow}e)$. □

We use this lemma to show:

Lemma 30 (Renaming lemma). $(\nu a)\,(\llbracket N\rrbracket a \mid a{\twoheadrightarrow}e) \quad \sim_{\mathrm{C}} \quad \llbracket N\rrbracket e$.

PROOF. By induction on the structure of terms in $\lambda\mathbf{x}$.

$$
\begin{array}{lll}
N = x: \quad (\nu a)\,(\llbracket x\rrbracket a \mid a{\twoheadrightarrow}e) & \triangleq & \\
\quad (\nu a)\,(x(z).z(w).\overline{a}\langle w\rangle \mid a(w).\overline{e}\langle w\rangle) & \sim_{\mathrm{C}} & \\
\quad x(z).z(w).\overline{e}\langle w\rangle & \triangleq & \llbracket x\rrbracket e
\end{array}
$$

$$
\begin{array}{lll}
N = \lambda x.M: \quad (\nu a)\,(\llbracket\lambda x.M\rrbracket a \mid a{\twoheadrightarrow}e) & \triangleq & \\
\quad (\nu a)\,((\nu x b)\,(\llbracket M\rrbracket b \mid \overline{a}\langle x,b\rangle) \mid a{\twoheadrightarrow}e) & \to_\pi & (a \notin \llbracket M\rrbracket b) \\
\quad (\nu x b)\,(\llbracket M\rrbracket b \mid \overline{e}\langle x,b\rangle) & \triangleq & \llbracket\lambda x.M\rrbracket e
\end{array}
$$

$$
\begin{array}{lll}
N = PQ: \quad (\nu a)\,(\llbracket PQ\rrbracket a \mid a{\twoheadrightarrow}e) & \triangleq & \\
\quad (\nu a)\,((\nu c)\,(\llbracket P\rrbracket c \mid c(b,d).(\llbracket b := Q\rrbracket \mid d{\twoheadrightarrow}a)) \mid a{\twoheadrightarrow}e) & \equiv & \\
\quad (\nu c)\,(\llbracket P\rrbracket c \mid (\nu a)\,(c(b,d).(\llbracket b := Q\rrbracket \mid d{\twoheadrightarrow}a) \mid a{\twoheadrightarrow}e)) & \sim_{\mathrm{C}} & (\textit{Lemma 29}) \\
\quad (\nu c)\,(\llbracket P\rrbracket c \mid c(b,d).(\nu a)\,(a{\twoheadrightarrow}e \mid \llbracket b := Q\rrbracket \mid d{\twoheadrightarrow}a)) & \sim_{\mathrm{C}} & \\
\quad (\nu c)\,(\llbracket P\rrbracket c \mid c(b,d).(\llbracket b := Q\rrbracket \mid d{\twoheadrightarrow}e)) & \triangleq & \llbracket PQ\rrbracket e
\end{array}
$$

$$
\begin{array}{lll}
N = M'\langle x := M\rangle: \quad (\nu a)\,(\llbracket M'\langle x := M\rangle\rrbracket a \mid a{\twoheadrightarrow}e) & \triangleq & \\
\quad (\nu a)\,((\nu x)\,(\llbracket M'\rrbracket a \mid \llbracket x := M\rrbracket) \mid a{\twoheadrightarrow}e) & \equiv & \\
\quad (\nu x)\,((\nu a)\,(\llbracket M'\rrbracket a \mid a{\twoheadrightarrow}e) \mid \llbracket x := M\rrbracket) & \sim_{\mathrm{C}} & (IH) \\
\quad (\nu x)\,(\llbracket M'\rrbracket e \mid \llbracket x := M\rrbracket) & \triangleq & \llbracket M'\langle x := M\rangle\rrbracket e
\end{array}
$$

□

Notice that, in the second part, reduction takes place over a private channel, so the processes involved are contextually equivalent.

In the previous lemma we have chosen the contextual equivalence to model the substitution, because this way of identifying programs is common for semantics of the λ-calculus as well. We could have chosen a co-inductive style equivalence such as the expansion [24] without changing Lemma 30 and what follows.

Using the laws of Lemma 30, we can show that:

$$\begin{array}{lll}(\nu xb)\,(\llbracket M\rrbracket b \mid !\,(\nu w)\,(\overline{x}\langle w\rangle.\,\llbracket N\rrbracket w) \mid b{\twoheadrightarrow}a) & \sim_{\mathrm{C}} & (\nu x)\,(\llbracket M\rrbracket a \mid !\,(\nu w)\,(\overline{x}\langle w\rangle.\,\llbracket N\rrbracket w)) \\ & \triangleq & (\nu x)\,(\llbracket M\rrbracket a \mid \llbracket x := N\rrbracket)\end{array}$$

Following on from Example 28, we can therefore justify:

Definition 31 (*Output*-based interpretation of $\lambda\mathbf{x}$ in π). We extend the interpretation of λ-terms in Definition 25 to $\lambda\mathbf{x}$ (and $\lambda\mathbf{x}_{\mathrm{L}}$ and $\lambda\mathbf{x}_{\mathrm{S}}$) by adding

$$\llbracket M\langle x := N\rangle\rrbracket a \quad = \quad (\nu x)\,(\llbracket M\rrbracket a \mid \llbracket x := N\rrbracket)$$

to our encoding.

As in [19, 24, 26], we can now show a reduction preservation result for explicit spine reduction. Notice that, essentially following Milner, by using the reduction relation $\rightarrow_{\mathbf{xs}}$, we show that our interpretation respects reduction in $\rightarrow_{\mathrm{S}}$ upto substitution, as expressed in Theorem 35. As in Theorem 18, we do not require the terms to be closed:

Theorem 32 ($\llbracket\cdot\rrbracket\cdot$ preserves $\rightarrow_{\mathbf{xs}}$ up to renaming). *If $M \rightarrow_{\mathbf{xs}} N$, then $\llbracket M\rrbracket a \sim_{\mathrm{C}} \llbracket N\rrbracket a$.*

PROOF. By induction on the definition of explicit spine reduction; we only show the basic cases.

$$\begin{array}{lll}(\lambda y.P)Q \rightarrow_{\mathbf{xs}} P\langle y := Q\rangle: \quad \llbracket(\lambda y.P)Q\rrbracket a & \triangleq & \\ \quad (\nu c)\,((\nu y b_1)\,(\llbracket P\rrbracket b_1 \mid \overline{c}\langle y, b_1\rangle) \mid c(b,d).\,(\llbracket b := Q\rrbracket \mid d{\twoheadrightarrow}a)) & \rightarrow_{\pi} & (28) \\ \quad (\nu y b_1)\,(\llbracket P\rrbracket b_1 \mid \llbracket y := Q\rrbracket \mid b_1{\twoheadrightarrow}a) & \sim_{\mathrm{C}} & (30) \\ \quad (\nu y)\,(\llbracket P\rrbracket a \mid \llbracket y := Q\rrbracket) & \triangleq & \llbracket P\langle y := Q\rangle\rrbracket a\end{array}$$

$$\begin{array}{lll}(\lambda y.M)\langle x := N\rangle \rightarrow_{\mathbf{xs}} \lambda y.(M\langle x := N\rangle): \quad \llbracket(\lambda y.M)\langle x := N\rangle\rrbracket a & \triangleq & \\ \quad (\nu x)\,((\nu y b)\,(\llbracket M\rrbracket b \mid \overline{a}\langle y, b\rangle) \mid \llbracket x := N\rrbracket) & \equiv & \\ \quad (\nu x y b)\,(\llbracket M\rrbracket b \mid \overline{a}\langle y, b\rangle \mid \llbracket x := N\rrbracket) & \equiv & \\ \quad (\nu y b)\,((\nu x)\,(\llbracket M\rrbracket b \mid \llbracket x := N\rrbracket) \mid \overline{a}\langle y, b\rangle) & \triangleq & \llbracket\lambda y.M\langle x := N\rangle\rrbracket a\end{array}$$

$$\begin{array}{lll}(xM_1\cdots M_n\,\overrightarrow{\langle y := L\rangle})\langle x := N\rangle \rightarrow_{\mathbf{xs}} (NM_1\cdots M_n\,\overrightarrow{\langle y := L\rangle})\langle x := N\rangle: & & \\ \quad \llbracket(xM_1\cdots M_n\,\overrightarrow{\langle y := L\rangle})\langle x := N\rangle\rrbracket a & \triangleq & \\ \quad (\nu x)\,(\llbracket xM_1\cdots M_n\,\overrightarrow{\langle y := L\rangle}\rrbracket a \mid \llbracket x := N\rrbracket) & \triangleq & \\ \quad (\nu x)\,((\nu c_n)\cdots((\nu c_2)\,((\nu c_1)\,(x(z).z(w).\overline{c_1}\langle w\rangle \mid c_1(b_1,d_1).\,(\llbracket b_1 := M_1\rrbracket \mid d_1{\twoheadrightarrow}c_2)) \mid & & \\ \qquad c_2(b_2,d_2).\,(\llbracket b_2 := M_2\rrbracket \mid d_2{\twoheadrightarrow}c_3)) \mid \cdots & & \\ \qquad\qquad c_n(b_n,d_n).\,(\llbracket b_n := M_n\rrbracket \mid d_n{\twoheadrightarrow}a)) \mid \overrightarrow{\llbracket y := L\rrbracket} \mid \llbracket x := N\rrbracket) & \equiv & \\ \quad (\nu x c_n\cdots c_2 c_1)\,(x(z_1).z_1(w).\overline{c_1}\langle w\rangle \mid c_1(b_1,d_1).\,(\llbracket b_1 := M_1\rrbracket \mid d_1{\twoheadrightarrow}c_2) \mid \cdots & & \\ \qquad c_n(b_n,d_n).\,(\llbracket b_n := M_n\rrbracket \mid d_n{\twoheadrightarrow}a) \mid \overrightarrow{\llbracket y := L\rrbracket} \mid (\nu v)\,(\overline{x}\langle v\rangle.\,\llbracket N\rrbracket v) \mid \llbracket x := N\rrbracket) & \rightarrow_{\pi} & (x) \\ \quad (\nu x c_n\cdots c_2 c_1 v)\,(v(w).\overline{c_1}\langle w\rangle \mid c_1(b_1,d_1).\,(\llbracket b_1 := M_1\rrbracket \mid d_1{\twoheadrightarrow}c_2) \mid \cdots & & \\ \qquad c_n(b_n,d_n).\,(\llbracket b_n := M_n\rrbracket \mid d_n{\twoheadrightarrow}a) \mid \overrightarrow{\llbracket y := L\rrbracket} \mid \llbracket N\rrbracket v \mid \llbracket x := N\rrbracket) & \sim_{\mathrm{C}} & (30) \\ \quad (\nu x c_n\cdots c_2 c_1)\,(\llbracket N\rrbracket c_1 \mid c_1(b_1,d_1).\,(\llbracket b_1 := M_1\rrbracket \mid d_1{\twoheadrightarrow}c_2) \mid \cdots & & \\ \qquad c_n(b_n,d_n).\,(\llbracket b_n := M_n\rrbracket \mid d_n{\twoheadrightarrow}a) \mid \overrightarrow{\llbracket y := L\rrbracket} \mid \llbracket x := N\rrbracket) & \equiv, \triangleq & \\ \quad \llbracket(NM_1\cdots M_n\,\overrightarrow{\langle y := L\rangle})\langle x := N\rangle\rrbracket a & & \end{array}$$

$$
\begin{array}{ll}
\llbracket(\lambda x.(\lambda z.(\lambda y.M)x))N\rrbracket a & \triangleq, \equiv \\
(\nu c x b_1)\,(\llbracket(\lambda z.(\lambda y.M)x)\rrbracket b_1 \mid \overline{c}\langle x, b_1\rangle \mid c(b,d).\,(\llbracket b := N\rrbracket \mid d \twoheadrightarrow a)) & \rightarrow_\pi (c) \\
(\nu x b_1)\,(\llbracket(\lambda z.(\lambda y.M)x)\rrbracket b_1 \mid \llbracket x := N\rrbracket \mid b_1 \twoheadrightarrow a) & \triangleq \\
(\nu x b_1)\,((\nu z b_2)(\llbracket(\lambda y.M)x\rrbracket b_2 \mid \overline{b_1}\langle z, b_2\rangle) \mid \llbracket x := N\rrbracket \mid b_1 \twoheadrightarrow a) & \triangleq \\
(\nu x b_1)\,((\nu z b_2)((\nu c_1)\,(\llbracket\lambda y.M\rrbracket c_1 \mid c_1(b,d).\,(\llbracket b := x\rrbracket \mid d \twoheadrightarrow b_2)) \mid \overline{b_1}\langle z, b_2\rangle) \mid \llbracket x := N\rrbracket \mid b_1 \twoheadrightarrow a) & \triangleq \\
(\nu x b_1)\,((\nu z b_2)((\nu c_1)\,((\nu y b_3)\,(\llbracket M\rrbracket b_3 \mid \overline{c_1}\langle y, b_3\rangle) \mid & \\
\qquad c_1(b,d).\,(\llbracket b := x\rrbracket \mid d \twoheadrightarrow b_2)) \mid \overline{b_1}\langle z, b_2\rangle) \mid \llbracket x := N\rrbracket \mid b_1 \twoheadrightarrow a) & \equiv \\
(\nu x b_1 z b_2 c_1 y b_3)\,(\llbracket M\rrbracket b_3 \mid \overline{c_1}\langle y, b_3\rangle \mid c_1(b,d).\,(\llbracket b := x\rrbracket \mid d \twoheadrightarrow b_2) \mid \overline{b_1}\langle z, b_2\rangle \mid \llbracket x := N\rrbracket \mid b_1 \twoheadrightarrow a) & \rightarrow_\pi (c_1) \\
(\nu x b_1 z b_2 y b_3)\,(\llbracket M\rrbracket b_3 \mid \llbracket y := x\rrbracket \mid b_3 \twoheadrightarrow b_2 \mid \overline{b_1}\langle z, b_2\rangle \mid \llbracket x := N\rrbracket \mid b_1 \twoheadrightarrow a) & \rightarrow_\pi (b_1) \\
(\nu x z b_2 y b_3)\,(\llbracket M\rrbracket b_3 \mid \llbracket y := x\rrbracket \mid b_3 \twoheadrightarrow b_2 \mid \overline{a}\langle z, b_2\rangle \mid \llbracket x := N\rrbracket) & \sim_{\mathrm{C}} (30) \\
(\nu x z b_2 y)\,(\llbracket M\rrbracket b_2 \mid \llbracket y := x\rrbracket \mid \overline{a}\langle z, b_2\rangle \mid \llbracket x := N\rrbracket) & \equiv \\
(\nu z b_2)\,((\nu x)\,((\nu y)\,(\llbracket M\rrbracket b_2 \mid \llbracket y := x\rrbracket) \mid \llbracket x := N\rrbracket) \mid \overline{a}\langle z, b_2\rangle) & \triangleq \\
(\nu z b_2)\,((\nu x)\,(\llbracket M\langle y := x\rangle\rrbracket b_2 \mid \llbracket x := N\rrbracket) \mid \overline{a}\langle z, b_2\rangle) & \triangleq \\
(\nu z b_2)\,(\llbracket M\langle y := x\rangle\langle x := N\rangle\rrbracket b_2 \mid \overline{a}\langle z, b_2\rangle) & \triangleq \\
\llbracket\lambda z.M\langle y := x\rangle\langle x := N\rangle\rrbracket a &
\end{array}
$$

Figure 3: An illustration to the encoding result.

$M \rightarrow_{\mathbf{xs}} M' \Rightarrow MN \rightarrow_{\mathbf{xs}} M'N$:
$$
\begin{array}{lll}
\llbracket MN\rrbracket a & \triangleq & \\
(\nu c)\,(\llbracket M\rrbracket c \mid c(b,d).\,(\llbracket b := N\rrbracket \mid d \twoheadrightarrow a)) & \sim_{\mathrm{C}} & (IH) \\
(\nu c)\,(\llbracket M'\rrbracket c \mid c(b,d).\,(\llbracket b := N\rrbracket \mid d \twoheadrightarrow a)) & \triangleq & \llbracket M'N\rrbracket a
\end{array}
$$

$M \rightarrow_{\mathbf{xs}} M' \Rightarrow \lambda y.M \rightarrow_{\mathbf{xs}} \lambda y.M'$: Since $\llbracket\lambda y.M\rrbracket a \triangleq (\nu y b)\,(\llbracket M\rrbracket b \mid \overline{a}\langle y, b\rangle)$, the result follows by induction.

$M \rightarrow_{\mathbf{xs}} M' \Rightarrow M\langle x := N\rangle \rightarrow_{\mathbf{xs}} M'\langle x := N\rangle$: Since $\llbracket M\langle x := N\rangle\rrbracket a \triangleq (\nu x)\,(\llbracket M\rrbracket a \mid \llbracket x := N\rrbracket)$, the result follows by induction. □

Notice that, as in the proof of Lemma 30, in the first and third part, reduction takes place over a private channel, so the processes involved are contextually equivalent. Notice also that the renaming reduction is crucial for the third case, where we have $(\nu v)\,(v(w).\overline{c_1}\langle w\rangle \mid \llbracket N\rrbracket v)$, which corresponds to $(\nu v)\,(v \twoheadrightarrow c_1 \mid \llbracket N\rrbracket v)$ and we want to yield $\llbracket N\rrbracket c_1$.

Remark 33. In the second case of the proof of the previous theorem, we observe that no reduction takes place in the encoding. This is due to a discrepancy of the semantics view of the substitution. In the reduction rule of $(\lambda y.M)\langle x := P\rangle \rightarrow_{\mathbf{xs}} \lambda y.(M\langle x := P\rangle)$, effectively we move the substitution $(M\langle x := P\rangle)$ inside the λ-abstraction. This could be regarded, to some extent, as an associativity rule being implemented. In fact, all we do is to move the parenthesis from the λ-abstraction to the body of the function. Since the abstraction is modelled with the composition operator, the substitution in π-calculus becomes a matter of associativity.

So, perhaps contrary to expectation, since abstraction is not encoded using *input*, we can without problem model reduction, modulo renaming, under a λ-abstraction. Notice that we strongly need the asynchronous character in the encoding of abstraction to achieve the representation of spine reduction: thanks to the fact that $\llbracket\lambda x.M\rrbracket a = \overline{a}\langle x, b\rangle \mid \llbracket M\rrbracket b$, the third part of the above proof is possible. This result is illustrated in Figure 3.

Example 34. As mentioned above, in $\rightarrow_{\mathbf{xs}}$ we can reduce as follows:

$$
\begin{array}{lll}
(\lambda x.xx)(\lambda y.y) \rightarrow_{\mathbf{xs}} & xx\langle x := (\lambda y.y)\rangle & \rightarrow_{\mathbf{xs}} \\
& ((\lambda y.y)x)\langle x := (\lambda y.y)\rangle & \rightarrow_{\mathbf{xs}} \\
& (y\langle y := x\rangle)\langle x := (\lambda y.y)\rangle & \rightarrow_{\mathbf{xs}} \\
& x\langle y := x\rangle\langle x := (\lambda y.y)\rangle & \rightarrow_{\mathbf{xs}} \quad (\lambda z.z)\langle y := x\rangle\langle x := (\lambda y.y)\rangle
\end{array}
$$

Then, by repeatedly applying Theorem 32, only

$$\begin{array}{lll} \llbracket(\lambda x.xx)(\lambda y.y)\rrbracket a & \sim_{\mathrm{C}} & \llbracket xx\langle x:=\lambda y.y\rangle\rrbracket a \quad \sim_{\mathrm{C}} \\ & & \llbracket(\lambda y.y)x\langle x:=\lambda y.y\rangle\rrbracket a \quad \sim_{\mathrm{C}} \\ & & \llbracket y\langle y:=x\rangle\langle x:=\lambda y.y\rangle\rrbracket a \quad \sim_{\mathrm{C}} \\ & & \llbracket x\langle y:=x\rangle\langle x:=\lambda y.y\rangle\rrbracket a \quad \sim_{\mathrm{C}} \quad \llbracket(\lambda z.z)\langle y:=x\rangle\langle x:=\lambda y.y\rangle\rrbracket a \end{array}$$

Notice that, because the encoding implements a limited notion of substitution, as for Milner's encoding, the reduction does *not* run past

$$(\nu c)\,(\llbracket\lambda y.y\rrbracket c \mid c(b,d).(\llbracket b:=\lambda y.y\rrbracket \mid d{\rightarrowtail}a)) \quad \triangleq \quad \llbracket(\lambda y.y)(\lambda y.y)\rrbracket a.$$

The only expression that gets close is that in the sixth line, which corresponds (up to renaming) to

$$(\nu x)\,((\nu c)\,(\llbracket\lambda y.y\rrbracket c \mid c(b,d).(\llbracket b:=x\rrbracket \mid d{\rightarrowtail}a)) \mid \llbracket x:=\lambda y.y\rrbracket) \quad \triangleq \quad \llbracket((\lambda y.y)x)\,\langle x:=\lambda y.y\rangle\rrbracket a$$

We can also show the following result.

Theorem 35 (Operational Soundness for explicit spine reduction). 1. *If $M \to^*_{\mathbf{x}\mathrm{S}} N$, then $\llbracket M\rrbracket a \sim_{\mathrm{C}} \llbracket N\rrbracket a$.*
2. *If $M\uparrow$ (*i.e. *all reduction paths in $\to_{\mathbf{x}\mathrm{S}}$ starting from M are of infinite length), then $\llbracket M\rrbracket a\uparrow$.*

PROOF. The first is shown by induction using Theorem 32; the second follows from the observation that an infinite reduction sequence in $\to_{\mathbf{x}\mathrm{S}}$ has infinitely many applications of rule (B), and from Example 28, each (B)-reduction step corresponds to at least one π-synchronisation step. □

Since lazy reduction is included in spine reduction, this immediately gives the following:

Corollary 36 (Operational Soundness for explicit lazy reduction). 1. *If $M \to^*_{\mathbf{x}\mathrm{L}} N$, then $\llbracket M\rrbracket a \sim_{\mathrm{C}} \llbracket N\rrbracket a$.*
2. *If $M\uparrow$ (with respect to $\to_{\mathbf{x}\mathrm{L}}$), then $\llbracket M\rrbracket a\uparrow$.*

We can even show a similar result for spine reduction:

Theorem 37 (Operational Soundness for spine reduction). 1. *If $M \to^*_{\mathrm{S}} N$, and M, N are pure terms, then there exists $P, \tilde{x}, \vec{Q}$ such that $\llbracket M\rrbracket a \sim_{\mathrm{C}} \llbracket P\langle\overline{x=Q}\rangle\rrbracket a$ and $P[\overrightarrow{Q/x}] = N$ (or $P\langle\overline{x=Q}\rangle \to_{:=} N$).*
2. *If $M\uparrow$ (with respect to $\to_{\mathrm{S}}$), then $\llbracket M\rrbracket a\uparrow$.*

PROOF. By Theorem 35, using Proposition 12. □

Of course we can state the same property for lazy reduction.

Note that this result is stronger than that for Milner's encoding (Theorem 16). Milner's encoding does not deal with step-by-step reduction, whereas we treat each individual reduction step in $\to_{\mathrm{S}}$.

By looking at the proof of Theorem 32 we can immediately deduce that $\llbracket\cdot\rrbracket\cdot$ preserves $=_{\mathbf{x}\mathrm{S}}$ up to $\sim_{\mathrm{C}}$, which states that our encoding gives, in fact, a semantics for the explicit substitution the λ-calculus: Milner's encoding does not deal with step-by-step reduction, and has its correctness expressed only through the applicative bisimulation, whereas we correctly treat each individual reduction step.

Corollary 38 (Adequacy). *If $M =_{\mathbf{x}\mathrm{S}} N$, then $\llbracket M\rrbracket a \sim_{\mathrm{C}} \llbracket N\rrbracket a$.*

This property also gives a proof for operational completeness for $\lambda\mathbf{x}\mathrm{S}$:

Theorem 39 (Operational completeness for $\lambda\mathbf{x}\mathrm{S}$). *If $\llbracket M\rrbracket a \to_\pi P$ then there exists $N \in \lambda\mathbf{x}$ such that $P \sim_{\mathrm{C}} \llbracket N\rrbracket a$, and $M \to^+_{\mathbf{x}\mathrm{S}} N$.*

PROOF. By easy induction on the structure of terms. □

$$
\begin{array}{lll}
\llbracket(\lambda x.xx)(\lambda y.y)\rrbracket a & \triangleq, \equiv & \\
(\nu cxb_1)\,(\llbracket xx\rrbracket b_1 \mid \overline{c}\langle x, b_1\rangle \mid c(b,d).\,(\llbracket b := \lambda y.y\rrbracket \mid d \twoheadrightarrow a)) & \to_\pi & (c) \\
(\nu xb_1)\,(\llbracket xx\rrbracket b_1 \mid \llbracket x := \lambda y.y\rrbracket \mid b_1 \twoheadrightarrow a) & \equiv, \triangleq & \\
(\nu xb_1)\,(\llbracket xx\rrbracket b_1 \mid (\nu v)\,(\overline{x}\langle v\rangle.\,(\nu yb)\,(\llbracket y\rrbracket b \mid \overline{v}\langle y, b\rangle)) \mid \llbracket x := \lambda y.y\rrbracket \mid b_1 \twoheadrightarrow a) & \triangleq, \equiv & \\
(\nu xb_1)\,((\nu c_1)\,(x(z).z(w).\overline{c_1}\langle w\rangle \mid c_1(b_2,d_2).\,(\llbracket b_2 := x\rrbracket \mid d_2 \twoheadrightarrow b_1)) \mid & & \\
\qquad (\nu v)\,(\overline{x}\langle v\rangle.\,(\nu yb)\,(\llbracket y\rrbracket b \mid \overline{v}\langle y, b\rangle)) \mid \llbracket x := \lambda y.y\rrbracket \mid b_1 \twoheadrightarrow a) & \to_\pi & (x, v) \\
(\nu xb_1)\,((\nu c_1)\,((\nu yb)\,(\overline{c_1}\langle y, b\rangle \mid c_1(b_2,d_2).\,(\llbracket b_2 := x\rrbracket \mid d_2 \twoheadrightarrow b_1) \mid \llbracket y\rrbracket b) \mid & & \\
\qquad \llbracket x := \lambda y.y\rrbracket \mid b_1 \twoheadrightarrow a)) & \to_\pi & (c_1) \\
(\nu xb_1)\,((\nu yb)\,(\llbracket y := x\rrbracket \mid b \twoheadrightarrow b_1 \mid \llbracket y\rrbracket b) \mid \llbracket x := \lambda y.y\rrbracket \mid b_1 \twoheadrightarrow a) & \equiv, \triangleq, =_\alpha & (u/y) \\
(\nu xb_1)\,((\nu yb)\,((\nu v)\,(\overline{y}\langle v\rangle.\,x(z).z(w).\overline{v}\langle w\rangle) \mid \llbracket y := x\rrbracket \mid b \twoheadrightarrow b_1 \mid y(z_1).z_1(w).\overline{b}\langle w\rangle) \mid & & \\
\qquad (\nu v_1)\,(\overline{x}\langle v_1\rangle.\,(\nu ub_2)\,(\llbracket u\rrbracket b_2 \mid \overline{v_1}\langle u, b_2\rangle)) \mid \llbracket x := \lambda y.y\rrbracket \mid b_1 \twoheadrightarrow a) & \to_\pi, \triangleq & (y) \\
(\nu xb_1v)\,((\nu b)\,(x(z).z(w).\overline{v}\langle w\rangle \mid (\nu y)\,(\llbracket y := x\rrbracket) \mid b \twoheadrightarrow b_1 \mid v(z).\overline{b}\langle z\rangle) \mid & & \\
\qquad (\nu v_1)\,(\overline{x}\langle v_1\rangle.\,(\nu ub_2)\,(\llbracket u\rrbracket b_2 \mid \overline{v_1}\langle u, b_2\rangle)) \mid \llbracket x := \lambda y.y\rrbracket \mid b_1 \twoheadrightarrow a) & \to_\pi, \triangleq & (x, v_1, v) \\
(\nu xb_1)\,((\nu bub_2)\,(\overline{b}\langle u, b_2\rangle \mid (\nu y)\,(\llbracket y := x\rrbracket) \mid b \twoheadrightarrow b_1) \mid \llbracket u\rrbracket b_2 \mid \llbracket x := \lambda y.y\rrbracket \mid b_1 \twoheadrightarrow a) & \to_\pi, \triangleq & (b, b_1) \\
\llbracket \lambda u.u\rrbracket a \mid (\nu x)\,((\nu y)\,(\llbracket y := x\rrbracket) \mid \llbracket x := \lambda y.y\rrbracket) & \sim_\pi & \llbracket \lambda u.u\rrbracket a
\end{array}
$$

Figure 4: Running $\llbracket(\lambda x.xx)(\lambda y.y)\rrbracket a$

We can even show that terminating $\to_{\mathbf{xS}}$ reductions correspond to terminating $\to_\pi$ reductions:

Theorem 40 (Termination). *If $M \to_{\mathbf{xS}}^* M'$, and M' is in normal form, then $\llbracket M\rrbracket a \sim_{\mathrm{C}} \llbracket M'\rrbracket a$, and the latter process cannot reduce.*

PROOF. If M' is in normal form with respect to $\to_{\mathbf{xS}}$, then it is of the shape $\lambda\vec{x}.zM_1\cdots M_n\,\overrightarrow{\langle y := L\rangle}$ with $z \notin \vec{y}$. Then, by Theorem 35, we have $\llbracket M\rrbracket a \sim_{\mathrm{C}} \llbracket M'\rrbracket a$. Notice that

$$
\begin{array}{ll}
\llbracket \lambda\vec{x}.zM_1\cdots M_n\,\overrightarrow{\langle y := L\rangle}\rrbracket a & = \\
(\nu x_1b_1)\,(\cdots(\nu x_nb_n)\,(\llbracket zM_1\cdots M_n\,\overrightarrow{\langle y := L\rangle}\rrbracket b_n \mid \overline{b_{n-1}}\langle x_n, b_n\rangle) \mid \cdots \mid \overline{a}\langle x_1, b_1\rangle) & = \\
(\nu x_1b_1)\,(\cdots(\nu x_nb_n)\,((\nu c_n)\cdots((\nu c_2)\,((\nu c_1)\,(\llbracket z\rrbracket c_1 \mid c_1(d_1,e_1).\,(\llbracket d_1 := M_1\rrbracket \mid e_1 \twoheadrightarrow c_2)) \mid & \\
\qquad c_2(d_2,d_2).\,(\llbracket d_2 := M_2\rrbracket \mid e_2 \twoheadrightarrow c_3)) \mid \cdots \mid c_n(d_n,d_n).\,(\llbracket d_n := M_n\rrbracket \mid e_n \twoheadrightarrow b_n)) \mid & \\
\qquad\qquad \overrightarrow{\llbracket y := L\rrbracket})) \mid \overline{b_{n-1}}\langle x_n, b_n\rangle) \mid \cdots \mid \overline{a}\langle x_1, b_1\rangle) &
\end{array}
$$

Now since $\llbracket v := N\rrbracket = !\,(\nu w)\,(\overline{v}\langle w\rangle.\,\llbracket N\rrbracket w)$, it is clear that, in this process, all possible synchronisations appear under *output*, so this process cannot reduce. □

6. Emulating Milner's reduction result

As is clear from the formulation of Corollary 36, we have modelled Milner's main result, as stated in Theorem 19, but not in full: notice that Milner's result maps a lazy λ-reduction path to a reduction in π, whereas Corollary 36 is formulated (in part) using contextual equivalence. This is due to the structure of the proof of Theorem 32, where this equivalence is used.

However, we can do one better, and show that we can emulate Milner's result through reduction only, i.e. do not need renaming to achieve this.

Example 41. We *can* run the π-process $\llbracket\lambda x.xx\rrbracket(\lambda y.y\rrbracket a$ without using renaming, as shown in Figure 4. Notice that there we perform the two substitutions without resorting to the renaming of *output*s of encoded λ-terms; these are executed after the encodings have participated in the execution.

In the proof of Theorem 32, in only two places do we perform a renaming (i.e. need the equivalence via Lemma 30). Notice that if N reduces to an abstraction $\lambda z.N'$, then (without loss of generality)

$$\begin{array}{rcl}(\nu a)\,(a\twoheadrightarrow e \mid \llbracket N\rrbracket a) & \to_\pi^* & (\nu a)\,(a\twoheadrightarrow e \mid (\nu zb)\,(\llbracket N'\rrbracket b \mid \overline{a}\langle z,b\rangle)) \\ & \equiv & (\nu azb)\,(a\twoheadrightarrow e \mid \llbracket N'\rrbracket b \mid \overline{a}\langle z,b\rangle) \\ & \to_\pi & (\nu zb)\,(\llbracket N'\rrbracket b \mid \overline{e}\langle z,b\rangle)\end{array}$$

so the renaming gets executed explicitly. When performing an explicit lazy reduction on a closed term M, then either M is an abstraction, so in normal form, or a redex of the shape $(\lambda x.P)(\lambda y.Q)$. Now this latter term gets (without renaming) interpreted by:

$$\begin{array}{rcl}\llbracket(\lambda x.P)(\lambda y.Q)\rrbracket a & \triangleq & (\nu c)\,((\nu xb_1)\,(\llbracket P\rrbracket b_1 \mid \overline{c}\langle x,b_1\rangle) \mid c(b,d).\,(\llbracket b := \lambda y.Q\rrbracket \mid d\twoheadrightarrow a)) \\ & \to_\pi\,(c) & (\nu xb_1)\,(\llbracket P\rrbracket b_1 \mid b_1\twoheadrightarrow a \mid \llbracket b := \lambda y.Q\rrbracket)\end{array}$$

Now assume $P = xP_1\cdots P_n$, then this reduction continues as follows:

$$\begin{array}{ll}(\nu xb_1)\,(\llbracket xP_1\cdots P_n\rrbracket b_1 \mid b_1\twoheadrightarrow a \mid \llbracket b := \lambda y.Q\rrbracket) & \triangleq \\ (\nu xb_1)\,((\nu c_1)\,(\cdots(\nu c_n)\,(x(w).\overline{c_n}\langle w\rangle \mid c_n(b,d).\,(\llbracket b := P_1\rrbracket \mid d\twoheadrightarrow c_{n-1})) \mid \cdots \mid & \\ \qquad c_1(b,d).\,(\llbracket b := P_n\rrbracket \mid d\twoheadrightarrow b_1))\mid b_1\twoheadrightarrow a \mid \llbracket b := \lambda y.Q\rrbracket) & \equiv \\ (\nu xb_1)\,((\nu c_1)\,(\cdots(\nu c_n)\,(x(z).z(w).\overline{c_n}\langle w\rangle \mid c_n(b,d).\,(\llbracket b := P_1\rrbracket \mid d\twoheadrightarrow c_{n-1})) \mid \cdots \mid & \\ \quad c_1(b,d).\,(\llbracket b := P_n\rrbracket \mid d\twoheadrightarrow b_1))\mid b_1\twoheadrightarrow a \mid (\nu v)\,(\overline{x}\langle v\rangle.\,(\nu yb_2)\,(\llbracket Q\rrbracket b_2 \mid \overline{v}\langle y,b_2\rangle)) \mid \llbracket b := \lambda y.Q\rrbracket) & \to_\pi\,(x,v) \\ (\nu xb_1)\,((\nu c_1)\,(\cdots(\nu c_n)\,(c_n(b,d).\,(\llbracket b := P_1\rrbracket \mid d\twoheadrightarrow c_{n-1})) \mid \cdots \mid & \\ \qquad c_1(b,d).\,(\llbracket b := P_n\rrbracket \mid d\twoheadrightarrow b_1))\mid b_1\twoheadrightarrow a \mid (\nu yb_2)\,(\llbracket Q\rrbracket b_2 \mid \overline{c_n}\langle y,b_2\rangle) \mid \llbracket b := \lambda y.Q\rrbracket) & \equiv \\ (\nu xb_1)\,(\llbracket(\lambda y.Q)P_1\cdots P_n\rrbracket b_1 \mid b_1\twoheadrightarrow a \mid \llbracket b := \lambda y.Q\rrbracket) & \end{array}$$

without renaming. So, by the reasoning above, when simulating lazy explicit reduction, renamings can be postponed, and the equivalence relation is not needed.

This immediately gives that, as in Corollary 19, we can now state:

Corollary 42. *If M is closed, and $M \to_{\mathbf{x}_L} (\lambda y.N)\overrightarrow{\langle x := L\rangle}$, then $\llbracket M\rrbracket a \to_\pi^* \llbracket\lambda y.N\rrbracket a \mid !\,\overrightarrow{\llbracket x := N\rrbracket}$.*

which reproves Milner's result, but now using the logical encoding.

As an example where the renamings do *not* disappear, consider

$$\begin{array}{ll}\llbracket(\lambda x.xx)(\lambda x.xx)\rrbracket a & \triangleq \\ (\nu c)\,((\nu xb_1)\,(\llbracket xx\rrbracket b_1 \mid \overline{c}\langle x,b_1\rangle) \mid c(b,d).\,(\llbracket b := (\lambda x.xx)\rrbracket \mid d\twoheadrightarrow a)) & \to_\pi\ (c) \\ (\nu xb_1)\,(\llbracket xx\rrbracket b_1 \mid \llbracket x := \lambda y.yy\rrbracket \mid b_1\twoheadrightarrow a) & \triangleq \\ (\nu xb_1)\,((\nu c)\,(x(z).z(w).\overline{c}\langle w\rangle \mid c(b_2,d).\,(\llbracket b_2 := x\rrbracket \mid d\twoheadrightarrow b_1)) \mid & \\ \qquad (\nu v)\,(\overline{x}\langle v\rangle.\,(\nu yb)\,(\llbracket yy\rrbracket b \mid \overline{v}\langle y,b\rangle)) \mid \llbracket x := \lambda y.yy\rrbracket \mid b_1\twoheadrightarrow a) & \to_\pi\ (x,v,c) \\ (\nu xb_1)\,((\nu yb)\,(\llbracket y := x\rrbracket \mid b\twoheadrightarrow b_1 \mid \llbracket yy\rrbracket b) \mid \llbracket x := \lambda y.y\rrbracket \mid b_1\twoheadrightarrow a) & \triangleq,\equiv\,(c) \\ (\nu xb_1)\,((\nu yb)\,(x(z).z(w).\overline{y}\langle w\rangle \mid \llbracket y := x\rrbracket \mid b\twoheadrightarrow b_1 \mid \llbracket yy\rrbracket b) \mid \llbracket x := \lambda y.y\rrbracket \mid b_1\twoheadrightarrow a) & \end{array}$$

Notice that continuing this reduction will communicate via the first y in $\llbracket yy\rrbracket b$, where

$$\llbracket yy\rrbracket b \quad \triangleq \quad (\nu c)\,(y(z).z(w).\overline{c}\langle w\rangle \mid c(b,d).\,(\llbracket b := y\rrbracket \mid d\twoheadrightarrow b))$$

not the second, and the *output* hidden in that term via b will never be performed, and neither will the *output* via b_1 or a; see also Example 11 (2).

7. Context assignment

The π-calculus is equipped with a rich type theory [26]: from the basic type system for counting the arity of channels, via a systems that registers the *input-output* use of channel names that are transmitted in [22], to sophisticated linear types in [17], which studies a relation between Call-by-Value $\lambda\mu$ and a linear π-calculus. Linearisation is used to be able to achieve processes that are functions, by allowing *output* over one channel name only, in a (λ-calculus) natural deduction style. Moreover, the encoding presented in [17] is type dependent, in that, for each term, different π-processes are assigned, depending on the original type; this makes the encoding quite cumbersome.

The notion of context assignment for processes in π we define in this section was first presented in [4] and differs quite drastically from the standard type system presented in [26]. It describes the '*input-output interface*' of a process by assigning a left context, containing the types for the *input* channels, and a right context, containing the types for the *output* channels; this implies that, if a name is both used to send and to receive, it will appear on both sides, and with the same type. In our system, types give a logical view to the π-calculus rather than an abstract specification on how channels should behave, and *input* and *output* channels essentially have the type of the data they are sending or receiving.

Context assignment was defined in [4] to establish preservation of assignable types under the interpretation of the sequent calculus $\mathcal{X}$, as presented in [6], into the π-calculus. Since $\mathcal{X}$ offers a natural presentation of the classical propositional calculus with implication, and enjoys the Curry-Howard isomorphism for the implicative fragment of Gentzen's system LK [12], this implies that the notion of context assignment as defined below is *classical* (i.e. not intuitionistic) in nature.

We now repeat the definition of (simple) type assignment; we first define types and contexts.

Definition 43 (Types and Contexts).

1. The set of types is defined by the grammar:

$$A, B \quad ::= \quad \varphi \mid A \to B$$

   where φ is a basic type of which there are infinitely many. The types considered in this paper are normally known as *simple* (or *Curry*) types.
2. An *input context* Γ is a mapping from names to types, denoted as a finite set of *statements* $n{:}A$, such that the *subject* of the statements (n) are distinct. We write Γ_1, Γ_2 to mean the *compatible union* of Γ_1 and Γ_2 (if Γ_1 contains $n{:}A_1$ and Γ_2 contains $n{:}A_2$, then $A_1 = A_2$), and write $\Gamma, n{:}A$ for $\Gamma, \{n{:}A\}$.
3. *Output* contexts Δ, and the notions Δ_1, Δ_2, and $n{:}A, \Delta$ are defined in a similar way.
4. If $n{:}A \in \Gamma$ and $n{:}B \in \Delta$, then $A = B$.

So, when writing a context as $\Gamma, n{:}A$, this implies that $n{:}A \in \Gamma$, or Γ is not defined on n.

Definition 44 ((Classical) Context Assignment). Context assignment for the π-calculus with pairing is defined by the following sequent system:

$$(0): \ \frac{}{0 : \Gamma \vdash_\pi \Delta} \qquad (!): \ \frac{P : \Gamma \vdash_\pi \Delta}{!\,P : \Gamma \vdash_\pi \Delta} \qquad (\textit{pair-out}): \ \frac{P : \Gamma, b{:}A \vdash_\pi c{:}B, \Delta}{\overline{a}\langle b, c\rangle . P : \Gamma, b{:}A \vdash_\pi a{:}A{\to}B, c{:}B, \Delta}\ (b \notin \Delta; a, c \notin \Gamma)$$

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

$$(|): \ \frac{P_1 : \Gamma \vdash_\pi \Delta \quad \cdots \quad P_n : \Gamma \vdash_\pi \Delta}{P_1 \mid \cdots \mid P_n : \Gamma \vdash_\pi \Delta} \qquad (\textit{let}): \ \frac{P : \Gamma, y{:}B \vdash_\pi x{:}A, \Delta}{\textit{let}\,\langle x, y\rangle = z\ \textit{in}\ P : \Gamma, z{:}A{\to}B \vdash_\pi \Delta}\ (y, z \notin \Delta; x \notin \Gamma)$$

$$(W): \ \frac{P : \Gamma \vdash_\pi \Delta}{P : \Gamma' \vdash_\pi \Delta'}\ (\Gamma' \supseteq \Gamma, \Delta' \supseteq \Delta) \qquad (\textit{in}): \ \frac{P : \Gamma, x{:}A \vdash_\pi x{:}A.\Delta}{a(x) . P : \Gamma, a{:}A \vdash_\pi \Delta}$$

As usual, we write $P : \Gamma \vdash_\pi \Delta$ if there exists a derivation using these rules that has this expression in the conclusion, and write $\mathcal{D} :: P : \Gamma \vdash_\pi \Delta$ if we want to name that derivation.

The side-condition on rule (out) is there to block the derivation of $\overline{a}\langle a\rangle : \vdash_\pi a{:}A$.

Notice that the above system is not trivial, since the process

$$(\nu cb)\,(x(w).\overline{c}\langle w\rangle \mid c(v,d).(!\,b\rightarrow v \mid d\rightarrow a) \mid !\,x(w_1).\overline{b}\langle w_1\rangle)$$

is not typeable: the left-hand x would need the type $A\rightarrow B$, and the right-hand x the type A.

Example 45. Although we have no rule $(pair\text{-}in)$, it is admissible, since we can derive

$$\dfrac{\dfrac{\boxed{\phantom{P : \Gamma, y{:}B \vdash_\pi x{:}A, \Delta}}}{let\,\langle x,y\rangle = z\ in\ P : \Gamma, z{:}A\rightarrow B \vdash_\pi \Delta}\,(let)}{a(z).\,let\,\langle x,y\rangle = z\ in\ P : \Gamma, a{:}A\rightarrow B \vdash_\pi \Delta}\,(in)$$

so the following rule is derivable:

$$(pair\text{-}in): \quad \dfrac{P : \Gamma, y{:}B \vdash_\pi x{:}A, \Delta}{a(x,y).\,P : \Gamma, a{:}A\rightarrow B \vdash_\pi \Delta}$$

This notion of type assignment does not (directly) relate back to the logical calculus LK. For example, rules $(|)$ and $(!)$ do not change the contexts, so do not correspond to any rule in LK, not even to a $\lambda\mu$-style [20] activation step; moreover, rule (ν) just removes a formula.

The weakening rule allows us to be a little less precise when we construct derivations, and allow for rules to join contexts, by using, for example, the rule

$$(|): \quad \dfrac{P : \Gamma_1 \vdash_\pi \Delta_1 \quad Q : \Gamma_2 \vdash_\pi \Delta_2}{P \mid Q : \Gamma_1, \Gamma_2 \vdash_\pi \Delta_1, \Delta_2}$$

so switching, without any scruples, to multiplicative style, whenever convenient. We will also write

$$\overline{\overline{a}\langle x,y\rangle : x{:}A \vdash_\pi a{:}A\rightarrow B, y{:}B}$$

instead of

$$\dfrac{\overline{\mathbf{0} : x{:}A \vdash_\pi y{:}B}}{\overline{a}\langle x,y\rangle.\,\mathbf{0} : x{:}A \vdash_\pi a{:}A\rightarrow B, y{:}B}$$

We have a soundness (witness reduction) result for our notion of type assignment for π as shown in [4].

Theorem 46 (Witness reduction [4]). *If $P : \Gamma \vdash_\pi \Delta$ and $P \rightarrow_\pi Q$, then $Q : \Gamma \vdash_\pi \Delta$.*

We will now show that our interpretation preserves types assignable to λ-terms using Curry's system, which is defined as follows:

Definition 47 (Curry type assignment for the λ-calculus). Curry type assignment is defined through the following inference rules:

$$(Ax): \quad \overline{\Gamma, x{:}A \vdash_\lambda x : A} \qquad (\rightarrow I): \quad \dfrac{\Gamma, x{:}A \vdash_\lambda M : B}{\Gamma \vdash_\lambda \lambda x.M : A\rightarrow B} \qquad (\rightarrow E): \quad \dfrac{\Gamma \vdash_\lambda M : A\rightarrow B \quad \Gamma \vdash_\lambda N : A}{\Gamma \vdash_\lambda MN : B}$$

Type preservation via $\llbracket\cdot\rrbracket\cdot$ is expressed by:

Theorem 48. *If $\Gamma \vdash_\lambda M : A$, then $\llbracket M\rrbracket a : \Gamma \vdash_\pi a{:}A$.*

PROOF. By induction on the structure of derivations in $\vdash_\lambda$; notice that we use implicit weakening.

(Ax) : Then $M = x$, and $\Gamma = \Gamma', x{:}A$. Notice that $x(z).z(w).\overline{a}\langle w\rangle = \llbracket x \rrbracket a$, and that[15]

$$\dfrac{\dfrac{\dfrac{}{\overline{a}\langle w\rangle : \Gamma', w{:}A \vdash_\pi a{:}A, w{:}A}\,(out)}{z(w).\overline{a}\langle w\rangle : \Gamma', z{:}A \vdash_\pi a{:}A}\,(in)}{x(z).z(w).\overline{a}\langle w\rangle : \Gamma', x{:}A \vdash_\pi a{:}A}\,(in)$$

$(\to I)$: Then $M = \lambda x.N$, $A = C{\to}D$, and $\Gamma, x{:}C \vdash_\lambda N : D$. Then, by induction, $\mathcal{D} :: \llbracket N \rrbracket b : \Gamma, x{:}C \vdash_\pi b{:}D$ exists, and we can construct:

$$\dfrac{\dfrac{\dfrac{\dfrac{\boxed{\mathcal{D}}}{\llbracket N \rrbracket b : \Gamma, x{:}C \vdash_\pi b{:}D} \qquad \dfrac{}{\overline{a}\langle x, b\rangle : x{:}C \vdash_\pi a{:}C{\to}D, b{:}D}\,(pair\text{-}out)}{\llbracket N \rrbracket b \mid \overline{a}\langle x, b\rangle : \Gamma, x{:}C \vdash_\pi a{:}C{\to}D, b{:}D}\,(|)}{(\nu b)(\llbracket N \rrbracket b \mid \overline{a}\langle x, b\rangle) : \Gamma, x{:}C \vdash_\pi a{:}C{\to}D}\,(\nu)}{(\nu xb)\,(\llbracket N \rrbracket b \mid \overline{a}\langle x, b\rangle) : \Gamma \vdash_\pi a{:}C{\to}D}\,(\nu)$$

Notice that $(\nu xb)\,(\llbracket N \rrbracket b \mid \overline{a}\langle x, b\rangle) = \llbracket \lambda x.N \rrbracket a$.

$(\to E)$: Then $M = PQ$, and there exists B such that $\Gamma \vdash_\lambda P : B{\to}A$ and $\Gamma \vdash_\lambda Q : B$. By induction, there exist derivations $\mathcal{D}_1 :: \llbracket P \rrbracket c : \Gamma \vdash_\pi c{:}B{\to}A$ and $\mathcal{D}_2 :: \llbracket Q \rrbracket w : \Gamma \vdash_\pi w{:}B$, and we can construct:

$$\dfrac{\dfrac{\boxed{\mathcal{D}_1}}{\llbracket P \rrbracket c : \Gamma \vdash_\pi c{:}B{\to}A} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\boxed{\mathcal{D}_2}}{\llbracket Q \rrbracket w : \Gamma \vdash_\pi w{:}B}}{\overline{b}\langle w\rangle.\llbracket Q \rrbracket w : \Gamma \vdash_\pi b{:}B, w{:}B}\,(out)}{(\nu w)\,(\overline{b}\langle w\rangle.\llbracket Q \rrbracket w) : \Gamma \vdash_\pi b{:}B}\,(\nu)}{!\,(\nu w)\,(\overline{b}\langle w\rangle.\llbracket Q \rrbracket w) : \Gamma \vdash_\pi b{:}B}\,(!) \qquad \dfrac{\dfrac{}{\overline{a}\langle w\rangle : w{:}A \vdash_\pi a{:}A, w{:}A}\,(out)}{d{\twoheadrightarrow}a : d{:}A \vdash_\pi a{:}A}\,(in)}{!\,(\nu w)\,(\overline{b}\langle w\rangle.\llbracket Q \rrbracket w) \mid d{\twoheadrightarrow}a : \Gamma, d{:}A \vdash_\pi a{:}A, b{:}B}\,(|)}{c(b,d).(!\,(\nu w)\,(\overline{b}\langle w\rangle.\llbracket Q \rrbracket w) \mid d{\twoheadrightarrow}a) : \Gamma, c{:}B{\to}A \vdash_\pi a{:}A}\,(pair\text{-}in)}{\llbracket P \rrbracket c \mid c(b,d).(!\,(\nu w)\,(\overline{b}\langle w\rangle.\llbracket Q \rrbracket w)) \mid d{\twoheadrightarrow}a) : \Gamma, c{:}B{\to}A \vdash_\pi c{:}B{\to}A, a{:}A}\,(|)}{(\nu c)\,(\llbracket P \rrbracket c \mid c(b,d).(\llbracket b := Q \rrbracket \mid d{\twoheadrightarrow}a)) : \Gamma \vdash_\pi a{:}A}\,(\nu)$$

and $(\nu c)\,(\llbracket P \rrbracket c \mid c(b,d).(\llbracket b := Q \rrbracket \mid d{\twoheadrightarrow}a)) = \llbracket PQ \rrbracket a$. □

Notice that although, in the above proof, we are only interested in showing results with *one* typed *output* (conclusion) – after all, we are interpreting the typed the λ-calculus, an intuitionistic system – we need the classical, multi-conclusion character of our type assignment system for π to achieve this result.

[15] It might be tempting to see the type assignment system in the view of the traditional systems, where types contain channel information, and, for example, use the rules

$$(out): \quad \dfrac{P : \Gamma, b{:}A \vdash_\pi b{:}A, \Delta}{\overline{a}\langle b\rangle.P : \Gamma, b{:}A \vdash_\pi a{:}[A], b{:}A, \Delta}\,(a \neq b) \qquad (in): \quad \dfrac{P : \Gamma, x{:}A \vdash_\pi x{:}A.\Delta}{a(x).P : \Gamma, a{:}[A] \vdash_\pi \Delta}$$

This approach will certainly not work for $\llbracket \cdot \rrbracket \cdot$: notice that then we would derive

$$\dfrac{\dfrac{\dfrac{}{\overline{a}\langle w\rangle : \Gamma', w{:}A \vdash_\pi a{:}A, w{:}A}\,(out)}{z(w).\overline{a}\langle w\rangle : \Gamma', z{:}A \vdash_\pi a{:}A}\,(in)}{x(z).z(w).\overline{a}\langle w\rangle : \Gamma', x{:}[A] \vdash_\pi a{:}A}\,(in)$$

destroying the preservation of assignable types; in the third case of the proof, we would now derive the judgement $!\,(\nu w)\,(\overline{b}\langle w\rangle.\llbracket Q \rrbracket w) : \Gamma \vdash_\pi b{:}[B]$, disrupting that derivation as well.

A natural question to ask is if also the processes created by Milner's encoding are typeable in $\vdash_\pi$. To investigate this question, essentially following Definition 13, we first define an *input*-based encoding of λ-terms - a variant of Milner's encoding - into the synchronous π-calculus with pairing by:

$$\begin{array}{rcll}
[\![x]\!]^{\mathrm{P}} a & \triangleq & \overline{x}\langle a\rangle & x \neq a \\
[\![\lambda x.M]\!]^{\mathrm{P}} a & \triangleq & a(x,b)\,.\,[\![M]\!]^{\mathrm{P}} b & b \textit{ fresh} \\
[\![MN]\!]^{\mathrm{P}} a & \triangleq & (\nu c)\,([\![M]\!]^{\mathrm{P}} c \mid (\nu z)\,(\overline{c}\langle z,a\rangle\,.\,[\![z := N]\!]^{\mathrm{P}})) & c,z \textit{ fresh} \\
[\![\langle x := N\rangle]\!]^{\mathrm{P}} & \triangleq & !\,x(w)\,.\,[\![N]\!]^{\mathrm{P}} w & c,z \textit{ fresh}
\end{array}$$

(Notice the similarity with the uniform encoding $[\![\cdot]\!]^{\mathrm{U}}\cdot$ from Definition 21.) Remark that now the use of synchronous synchronisation, as evident in the third case, does not enlarge the expressiveness of the interpretation, since abstraction is still interpreted using *input*. Notice that this interpretation is well behaved: using $[\![\cdot]\!]^{\mathrm{P}}\cdot$, the interpretation of a β-redex reduces as follows:

$$\begin{array}{rcll}
[\![(\lambda x.M)N]\!]^{\mathrm{P}} a & \triangleq & (\nu c)\,(c(x,b)\,.\,[\![M]\!]^{\mathrm{P}} b \mid (\nu z)\,(\overline{c}\langle z,a\rangle\,.\,[\![z := N]\!]^{\mathrm{P}})) & \to_\pi \quad (c) \\
 & & (\nu z)\,([\![M[z/x]]\!]^{\mathrm{P}} a \mid [\![z := N]\!]^{\mathrm{P}}) & = \quad (z \notin [\![M]\!]^{\mathrm{P}} a) \\
 & & (\nu x)\,([\![M]\!]^{\mathrm{P}} a \mid [\![x := N]\!]^{\mathrm{P}}) &
\end{array}$$

exactly as expected.

Notice that, in this encoding, all the λ-calculus *input* variables are interpreted as *output* channels, and that a is the only *input* variable for each encoded λ-term; so we could hope to show " *If* $\Gamma \vdash_\lambda M : A$, *then* $[\![M]\!]^{\mathrm{P}} a : a{:}A \vdash_\pi \Gamma$ ". However, this is not possible; although it nicely holds for the first two cases of the proof, we cannot show it for the third case:

$M = PQ$: Then $\Gamma \vdash_\lambda P : B{\to}A$ and $\Gamma \vdash_\lambda Q : B$; then, by induction, we have derivations for $[\![P]\!]^{\mathrm{P}} c : c{:}B{\to}A \vdash_\pi \Gamma$ and $[\![Q]\!]^{\mathrm{P}} w : w{:}B \vdash_\pi \Gamma$, and we would like to construct:

$$\dfrac{\dfrac{\dfrac{\vdots}{[\![P]\!]^{\mathrm{P}} b : b{:}A \vdash_\pi x{:}B,\Gamma}}{c(x,b)\,.\,[\![P]\!]^{\mathrm{P}} b : c{:}B{\to}A \vdash_\pi \Gamma}\;(\textit{pair-in}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{[\![Q]\!]^{\mathrm{P}} w : a{:}A, w{:}B \vdash_\pi \Gamma}}{z(w)\,.\,[\![Q]\!]^{\mathrm{P}} w : z{:}B, a{:}A \vdash_\pi \Gamma}\;(\textit{in})}{!\,z(w)\,.\,[\![Q]\!]^{\mathrm{P}} w : z{:}B, a{:}A \vdash_\pi \Gamma}\;(!)}{\overline{c}\langle z,a\rangle\,.\,!\,z(w)\,.\,[\![Q]\!]^{\mathrm{P}} z : z{:}B, a{:}A \vdash_\pi c{:}A{\to}B,\Gamma}\;(??)}{(\nu z)\,(\overline{c}\langle a,z\rangle\,.\,!\,z(w)\,.\,[\![Q]\!]^{\mathrm{P}} z) : a{:}A \vdash_\pi c{:}A{\to}B,\Gamma}\;(\nu)}{\dfrac{c(x,b)\,.\,[\![P]\!]^{\mathrm{P}} b \mid (\nu z)\,(\overline{c}\langle z,a\rangle\,.\,!\,z(w)\,.\,[\![Q]\!]^{\mathrm{P}} z) : c{:}B{\to}A, a{:}A \vdash_\pi c{:}B{\to}A,\Gamma}{[\![(\lambda x.P)Q]\!]^{\mathrm{P}} a : a{:}A \vdash_\pi \Gamma}\;(\nu)}\;(\mid)$$

Notice that now rule $(\textit{pair-out})$ cannot be applied in position $(??)$, since it requires that the right-hand term in the pair is an *output*; so now a has to appear on the right of the turnstyle, which destroys the property we tried to prove.

Type assignment fails for the uniform encoding for the same reason.

Conclusions and Future Work

We have found a new, simple and intuitive encoding of λ-terms in π that respects our definition of explicit spine reduction, is similar with normal reduction, and encompasses Milner's lazy reduction on closed terms. We have shown that, for our context assignment system that uses the type constructor $\to$ for π and is based on classical logic, assignable types for λ-terms are preserved by our interpretation as typeable π-processes. We managed this without having to linearise the calculus as done in [17].

As we remarked in this paper, the guard we have placed on the encoding of the substitution is only there to guarantee the termination result; it plays no role in any of the other results we show. Even without that guard in place,

we can show the operational soundness result; this implies that eventual non-termination as a result of the unguarded replication is not observable. We aim to extend our results to an encoding that can represent *full step-by-step* β-reduction; we would have to drop the guard on replication to achieve that, but this in itself would not create problems with respect to the definition of semantics.

The classical sequent calculus $\mathcal{X}$ has two natural, dual notions of sub-reduction, called Call-by-Name and Call-by-Value; we will investigate if the interpretation of these systems in to the π-calculus gives natural notions of CBN of CBV reduction on π-processes, and if this enables CBN or CBV logical encodings of the λ-calculus.

Acknowledgements

We would like to thank Fer-Jan de Vries, Jan Willem Klop, Vincent van Oostrom, Claudio Sacerdoti Coen and Davide Sangiorgi for useful discussions, comments and suggestions.

References

[1] M. Abadi and A. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proceedings of the Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[2] S. Abramsky. The lazy lambda calculus. In *Research topics in functional programming*, pages 65–116. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[3] S. Abramsky. Proofs as processes. *Theoretical Computer Science*, 135(1):5–9, 1994.

[4] S. van Bakel, L. Cardelli, and M.G. Vigliotti. From $\mathcal{X}$ to π; Representing the Classical Sequent Calculus in the π-calculus. In *Electronic Proceedings of International Workshop on Classical Logic and Computation 2008* (CL&C'08), *Reykjavik, Iceland*, 2008.

[5] S. van Bakel, S. Lengrand, and P. Lescanne. The language $\mathcal{X}$: circuits, computations and Classical Logic. In M. Coppo, Elena Lodi, and G. Michele Pinna, editors, *Proceedings of Ninth ItalI.Conference on Theoretical Computer Science* (ICTCS'05), *Siena, Italy*, volume 3701 of *Lecture Notes in Computer Science*, pages 81–96. Springer Verlag, 2005.

[6] S. van Bakel and P. Lescanne. Computation with Classical Sequents. *Mathematical Structures in Computer Science*, 18:555–609, 2008.

[7] S. van Bakel and M.G. Vigliotti. A logical interpretation of the λ-calculus into the π-calculus, preserving spine reduction and types. In M. Bravetti and G. Zavattaro, editors, *Proceedings of 20th International Conference on Concurrency Theory* (CONCUR'09), Bologna, Italy, volume 5710 of *Lecture Notes in Computer Science*, pages 84 – 98. Springer Verlag, 2009.

[8] H. Barendregt. *The Lambda Calculus: its Syntax and Semantics*. North-Holland, Amsterdam, revised edition, 1984.

[9] H.P. Barendregt, R. Kennaway, J.W. Klop, and M.R. Sleep. Needed Reduction and Spine Strategies for the Lambda Calculus. *Information and Computation*, 75(3):191–231, 1987.

[10] G. Bellin and P.J. Scott. On the pi-Calculus and Linear Logic. *Theoretical Computer Science*, 135(1):11–65, 1994.

[11] R. Bloo and K.H. Rose. Preservation of Strong Normalisation in Named Lambda Calculi with Explicit Substitution and Garbage Collection. In *CSN'95 – Computer Science in the Netherlands*, pages 62–72, 1995.

[12] G. Gentzen. Investigations into logical deduction. In *The Collected Papers of Gerhard Gentzen*. Ed M. E. Szabo, North Holland, 68ff (1969), 1935.

[13] G. Gentzen. Untersuchungen über das Logische Schliessen. *Mathematische Zeitschrift*, 39:176–210 and 405–431, 1935.

[14] J. Goubault-Larrecq. A Few Remarks on SKInT. Research Report RR-3475, INRIA Rocquencourt, France, 1998.

[15] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *Proceedings of ECOOP'91*, volume 512 of *Lecture Notes in Computer Science*, pages 133–147. Springer Verlag, 1991.

[16] K. Honda and N. Yoshida. On the reduction-based process semantics. *Theoretical Computer Science*, 151:437–486, 1995.

[17] K. Honda, N. Yoshida, and M. Berger. Control in the π-Calculus. In *Proceedings of Fourth ACM-SIGPLAN Continuation Workshop* (CW'04), 2004.

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

[19] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):269–310, 1992.

[20] M. Parigot. An algorithmic interpretation of classical natural deduction. In *Proceedings of 3rd International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'92)*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer Verlag, 1992.

[21] J. Parrow and B. Victor. The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes. In *Proceedings of of 13th Annual IEEE Symposium on Principles on Logic in Computer Science*, pages 428–440, 1998.

[22] B.C. Pierce and D. Sangiorgi. Typing and Subtyping for Mobile Processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996.

[23] D. Sangiorgi. *Expressing Mobility in Process Algebra: First Order and Higher Order Paradigms*. PhD thesis, Edinburgh University, 1992.

[24] D. Sangiorgi. An Investigation into Functions as Processes. In *Proceedings of Mathematical Foundations of Programming Semantics, 9th International Conference, New Orleans, LA, USA*, pages 143–159, 1993.

[25] D. Sangiorgi. Lazy functions and mobile processes. Rapport de Recherche 2515, INRIA, Sophia-Antipolis, France, 1995.

[26] D. Sangiorgi and D. Walker. *The Pi-Calculus*. Cambridge University Press, 2001.

[27] P. Sestoft. Standard ML on the Web server. Department of Mathematics and Physics, Royal Veterinary and Agricultural University, Denmark, 1996.

[28] H. Thielecke. *Categorical Structure of Continuation Passing Style*. PhD thesis, University of Edinburgh, 1997. LFCS technical report ECS-LFCS-97-376.

[29] C. Urban. *Classical Logic and Computation*. PhD thesis, University of Cambridge, October 2000.
[30] C. Urban and G.M. Bierman. Strong normalisation of cut-elimination in classical logic. *Fundamenta Informaticae*, 45(1,2):123–155, 2001.
[31] F.-J. de Vries. Böhm trees, bisimulations and observations in lambda calculus. In T. Ida, A. Ohori, and M. Takeichi, editors, *Second Fuji International Workshop on Functional and Logic Programming Workshop, World Scientific, Singapore*, pages 230–245, 1997.