# Implicative Logic based translations of the $\lambda$-calculus into the $\pi$-calculus

Steffen van Bakel and Maria Grazia Vigliotti

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

**Abstract**

We study an *output*-based translation of the $\lambda$-calculus with explicit substitution into the synchronous $\pi$-calculus – enriched with pairing – that has its origin in mathematical logic, and show that this translation respects reduction. We will define the notion of (explicit) head reduction -which encompasses (explicit) lazy reduction- and show that the translation fully represents this reduction in that term-substitution as well as each single reduction step are modelled.

We show that all the main properties (soundness, completeness, and adequacy) hold for these notions of reduction, as well as that termination is preserved with respect to a notion of call by need reduction for the $\pi$-calculus. We then define a notion of type assignment for the $\pi$-calculus that uses the type constructor $\rightarrow$, and show that all Curry types assignable to $\lambda$-terms are preserved by the translation.

We will also show that the $\pi$-calculus gives a semantics for the (standard) $\lambda$-calculus by defining an encoding that will fully represent reduction with explicit substitution, $\beta$-reduction, and equality, mapping equivalent term to equivalent processes.

**keywords:** $\lambda$-calculus, $\pi$-calculus, intuitionistic logic, classical logic, translation, type assignment

## 1 Introduction

The $\pi$-calculus and its dialects have proven to be an interesting model of computation. Encoding of variants of pure [25, 32, 9, 31] and typed [22] $\lambda$-calculus [14, 10] and object oriented calculi [20, 32] have been defined. Also, various encodings of calculi that represent classical logic have been recently proposed [22, 5, 15].

In this paper we cut an alternative path in the investigation of the expressive power of the $\pi$-calculus [25] by presenting two new compositional encodings of $\lambda\mathbf{x}$, the explicit substitution $\lambda$-calculus [1, 13] into the asynchronous $\pi$-calculus extended with pairing. The first will be shown to fully respect *explicit head-reduction*, which corresponds closely to Krivine's machine [23]; the second will be shown to respect *full single-step reduction*. Both these notions extend *lazy* reduction [3] by allowing reduction also under $\lambda$-abstraction; the second will model also reduction inside the right-hand side of an application and inside the substitution. The great advantage of considering the explicit substitution $\lambda$-calculus rather than the standard $\lambda$-calculus with implicit substitution as considered in [25, 32] will become clear in this paper. In brief, the specifics of *synchronisation* – the main computational step in the standard (*i.e.* not *higher order*) $\pi$-calculus – is the reason that implicit substitution cannot be faithfully modelled. Rather, substitution is implemented in the $\pi$-calculus as '*one variable at the time*'; this implies that the kind of substitution modelled is, in fact, that of $\lambda\mathbf{x}$, where precise control over substitution is gained by making the substitution operation be part of the syntax and reduction system. This point was already argued in [9]; in that paper we showed that synchronisation in the $\pi$-calculus has a fine semantic level of granularity that 'faithfully mimics' explicit substitution, rather than implicit substitution; in fact, we showed

there that the nature of the $\pi$-calculus brings that an implementation of $\lambda$'s reduction comes down to what we here call *explicit head reduction*.

Research in the direction of translations of $\lambda$-terms was started by Milner in [25]; he defined an *input*-based encoding, and showed that the interpretation of closed $\lambda$-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 [32], who also investigated it in the context of the higher-order $\pi$-calculus; by Honda *et al.* [22] with a rich type theory; and by Thielecke [35] 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 translation of the $\lambda$-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. In view of its impact, perhaps surprisingly Milner's encoding does not respect *step-by-step* lazy $\beta$-reduction with implicit substitution, as we first argued in [9]; in fact, Milner's result is formulated through lazy large-step reduction, that reduces to lazy $\beta$-normal form directly. Also Sangiorgi [31] limits reduction to lazy large-step reduction; therefore, not all individual lazy reduction steps are modelled. We will address this point in this paper, by reformulating these result using explicit substitution.

The lazy reduction strategy is widely accepted in the context of functional programming and is the strategy of choice in many, and the above results show therefore that the $\pi$-calculus is expressive enough to represent the functional paradigm. However, we can also ask whether it can successfully represent the $\lambda$-calculus itself. As mentioned by Sestoft [34] (referring to Church numerals):

> " an implementation of lambda calculus reduction must perform reductions under lambda abstractions. Otherwise, *add two two* would not reduce to *four* (...), which would disappoint students. "

Although other representations of numbers exist that avoid this problem, the issue Sestoft raises is relevant to deciding what is a proper encoding of the $\lambda$-calculus. Therefore, we can justifiable ask the question: is the expressive power of the $\pi$-calculus by nature limited to lazy reduction, or can we, for example, represent head reduction or even full $\beta$-reduction? We answer those questions in this paper in the positive and define alternative encodings that represents those reductions as well. We shall see in the course of this paper how our work on the $\lambda$-calculus with *explicit substitution* and a novel type system will shed new light on the aforementioned representation problem.

In this paper we are mainly interested in what exactly is the maximal variant of the $\lambda$-calculus that can be comfortably embedded into the pure $\pi$-calculus: we therefore do not consider the higher order $\pi$-calculus of [29], nor do we consider to extend the $\pi$-calculus by allowing reduction under guard or replication as in [22]. We will choose an approach alternative to Milner's, interpreting terms under *output* rather than under *input*, and encoding head reduction - which encompasses lazy reduction - in the process. We will then generalise that encoding to one that effectively represents full $\beta$-reduction; in particular, we can show that the substitution lemma can be modelled.

## Substitution

The first question we will address is:

*How can we faithfully model (implicit) substitution of the $\lambda$-calculus in the $\pi$-calculus.*

This question is relevant; for example, although Milner's encoding is formulated using implicit substitution (see Theorem 4.3), it does not truly model that (as illustrated in Example 4.4).

We see two ways to address the problem of representing implicit $\lambda$-substitution in the $\pi$-calculus:

*i*) To encode the $\lambda$-calculus into the Higher-Order $\pi$-calculus, as done by Sangiorgi [30];

*ii*) To consider a different kind of $\lambda$-calculus where the substitution is more 'fine-grained'.

In this paper, we have chosen the second option; 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 $\lambda$-calculus, and the simulation of this operation in the $\pi$-calculus via channel-name passing.

Although both the $\lambda$-calculus and the $\pi$-calculus are equipped with substitution, these notions are conceptually very different. While the $\lambda$-calculus has an intrinsic '*high-order*' substitution mechanism by which terms get substituted for variables '*all in one go*', in the standard $\pi$-calculus the substitution mechanism replaces variables by channel names only, *i.e.* not by processes. Because of this discrepancy, it is not possible to assume that substitution of the $\lambda$-calculus can be translated straightforwardly into the same mechanism in the $\pi$-calculus.

We will see that $M[N/x]$ has to be modelled using replication on $N$. Since copies of replicated processes are extracted through the congruence $!P \equiv P \mid !P$ and there is no restriction on the congruence, the replicated substitution will always be present during the computation of the process generated by the translation, creating possibly undesired computations. This is apparent in Milner's encoding [25] (see also the formulation of Milner's result Theorem 4.3, Theorem 4.6, and Example 4.4. 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.

This 'persistence' of the substitution in the $\pi$-calculus is absent in the standard $\lambda$-calculus, but is present in our version of the (lazy) $\lambda$-calculus with explicit substitution, which is called the *explicit head reduction* calculus. Explicit substitution is generally considered an implementation of the $\lambda$-calculus, where every 'atomic step' of the substitution is represented in the reduction relation; in explicit head reduction we restrict the applicability of substitution to head-variables only, leaving other occurrences untouched (we see this behaviour also in Krivine's machine). We will establish a correspondence between explicit head reduction and synchronisation in the $\pi$-calculus, through which we emphasise that substitution in the $\pi$-calculus is more 'fine-grained' than that in the $\lambda$-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 4.6), modelling individual *explicit lazy* reduction steps.

Our translation is *conceptually different* from Milner's, being an interpretation under *output* rather than under *input*, which is a essentially the translation presented in [9]. Our translation has its origin in Gentzen's encoding of natural deduction into the Sequent Calculus [17], and is therefore logical in nature. Another difference between our approach and Milner's is that we model abstraction via a process that uses asynchronous *output*; this will allow us to model reduction under $\lambda$-abstraction as well, *i.e.* our new translation respects not only (explicit) lazy reduction, but also the (larger) notion of *explicit head reduction*. Our translation is not the first one to do that; in [27] the translation of head reduction $\lambda$-calculus into

the Fusion-calculus is presented. To obtain the main result, [27] need not change Milner's encoding, but achieves it as a consequence of the symmetric substitution mechanism introduced into the Fusion-calculus. Moreover, their translation is not related to (classical) logic, as is ours.

Expanding on these results, we will also define an alternative translation for which we will show that *full and complete step-by-step explicit* $\lambda\mathbf{x}$-reduction can be faithfully encoded into the $\pi$-calculus as well. From the representation of $\lambda\mathbf{x}$ follows that we can model $\lambda$-calculus' $\beta$-equality in full as well. We achieve this by generalising and modifying the logical translation.

For our translation we do obtain a strong correspondence theorem between reductions, but we shall need the help of a restricted variant of contextual equivalence to match terms perfectly through renaming of bound output names (which corresponds to explicit $\alpha$-conversion) and garbage collection. Furthermore, our translation allows to establish a relationship between $\pi$-calculus and logic. In fact, the central idea behind this translation interprets a typeable redex (a logical cut) as a synchronisation, which is, as mentioned above, essentially based on Gentzen's translation of Natural Deduction into the Sequent Calculus. The idea of giving a computational interpretation of the cut as a synchronisation primitive is also used by [4] and [12]; in both papers, only a small fragment of Linear Logic was considered, and the translation between proofs and the $\pi$-calculus was left rather implicit.

In summary, we feel we have gained faithfulness and clarity in considering the translation from the $\lambda$-calculus with explicit substitution to the $\pi$-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, essenstiallty, is the minimal substitution needed to reach the head-normal form of a term, if it exists.

### Type system

The second question we address is:

> *How can we define a notion of functional (implicative) type assignment for the $\pi$-calculus that can be naturally linked to such a notion for the $\lambda$-calculus.*

We will show a type preservation result for our translation, using the type system as presented in [5]. This type system is different from standard type systems for $\pi$ as it does not contain any channel information and in that it expresses implication. It provides a logical view to the $\pi$-calculus, where $\pi$-processes can be witnesses of formulae that are provable (within the implicative fragment) in classical logic, as was shown in [5]; this implies that the $\pi$-calculus provides computational content to classical logic.

We will show in this paper that our translation preserves types assignable to $\lambda$-terms in Curry's system, which, through the Curry-Howard correspondence corresponds to a Natural Deduction system for Implicative Intuitionistic Logic. Through this type preservation result we show that our translation 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 [22] as their type system needs a linear restriction of the behaviour of the $\pi$-calculus to achieve a full abstraction result, as well as a typed language and a type-based interpretation, a double negation elimination translation of types, and allowing synchronisation in the $\pi$-calculus also under input and replication.

The results on the type system that we present here determines the choice of the $\pi$-calculus used for the translation: we use the asynchronous $\pi$-calculus enriched with *pairs* of names [2]. In principle, our translation could be adapted to the asynchronous monadic $\pi$-calculus, however we would not be able to achieve the preservation of assignable types. Our transla-

tion takes inspiration from, but it is a much improved version of, the translation of $\lambda$-terms in to the sequent calculus $\mathcal{X}$ [7, 8] – a first variant was defined by Urban [36, 37]; $\mathcal{X}$ is a sequent calculus that enjoys the Curry-Howard correspondence for Gentzen's LK [18] – and the translation of $\mathcal{X}$ into the $\pi$-calculus as defined in [5].

Our work therefore 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.

### Paper outline

In Section 2, we repeat the definition of the asynchronous $\pi$-calculus with pairing, and in Section 3 that of the $\lambda$-calculus, where we present the notion of explicit head reduction '$\rightarrow_{\mathrm{xH}}$' which takes a central role in this paper. In Section 4 we briefly discuss Milner's interpretation result for the lazy $\lambda$-calculus and establish a relation with explicit head reduction. Then, in Section 5, we will define a translation where terms are interpreted under *output* rather than *input* (as in Milner's), and show in Section 5.1 that $\rightarrow_{\mathrm{xH}}$ is respected by our interpretation, modulo renaming and garbage collection; in Section 5.4, we show that renaming is not needed when interpreting closed terms in the lazy $\lambda$-calculus. In Section 6 we give a notion of (type) context assignment on processes in $\pi$, and show that our interpretation preserves assignable Curry types, which is therefore *logical*. To conclude, in Section 7 we define an encoding that respects full $\beta$-reduction.

This paper is a modified and extended version of the paper that appeared as [9]; we have, in particular, corrected the definition of explicit head reduction, addressed the termination issue, and extended our results with those for the full encoding.

## 2 The asynchronous $\pi$-calculus with pairing

The notion of asynchronous $\pi$-calculus that we consider in this paper is the one used also in [2, 5], and is different from other systems studied in the literature [20] in that 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 [2] lies in the fact that we want to preserve implicative type assignment.

The $\pi$-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 $\Gamma$ and $\Delta$ 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. Now if we want to model this in $\pi$, *i.e.* want to express *function construction* (or *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 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 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 2.1** (Processes) *i*) *Channel names* and *data* are defined by:

$$\begin{array}{rcll} & & a,b,c,d,x,y,z & \textit{names} \\ p & ::= & a \mid \langle a,b \rangle & \textit{data} \end{array}$$

Notice that pairing is *not* recursive.

*ii*) Processes are defined by:

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

*iii*) As usual, $\nu$ is a binder, and the name $n$ is *bound* in $(\nu n)\,P$; $n$ is *free* in $P$ if it occurs in $P$, but is not bound.

*iv*) We call a process *mute* if it has no free output names.

*v*) A *context* $\mathtt{C}[\cdot]$ is a process with a hole $[\ ]$.

Whether or not a process is mute is decidable, and preserved by internal reduction.

We consider processes modulo $\alpha$-conversion, and abbreviate $a(x).\,\mathit{let}\,\langle y,z \rangle = x\ \mathit{in}\ P$ by $a(y,z).\,P$, and $(\nu m)\,(\nu n)\,P$ by $(\nu mn)\,P$, and write $\overline{a}\langle c,d \rangle$ for $\overline{a}\langle\langle c,d \rangle\rangle$. 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 [21]), which for convenience is abbreviated into $a \rightarrow b$.

It is worthwhile to point out that using pairing is not the same as working with the polyadic (or even dyadic) $\pi$-calculus, because in that setting each channel has a fixed arity, whereas we allow data to be sent, which is either a name or a pair of names. This implies that a process like $\overline{a}\langle b,c \rangle.P \mid a(x).\,Q$ can synchronise over $a$, but also that $\mathit{let}\,\langle x,y \rangle = a\ \mathit{in}\ P$ is stuck.

**Definition 2.2** (Congruence) The structural congruence is the smallest equivalence relation closed under contexts defined by the following rules:

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

We will consider processes modulo congruence: this implies that we will not deal explicitly with the process $\mathit{let}\,\langle x,y \rangle = \langle a,b \rangle\ \mathit{in}\ P$, but rather with $P[a/x,b/y]$. Because of the third rule we will normally not write brackets in a parallel composition of more than two processes. Also, through structural congruence, we can consider $!P$ a countable infinite repetition of $P$: take $!_1 P \triangleq P$, and $!_{n+1} P \triangleq P \mid !_n P$, then $!P \triangleq !_\omega P$ (where $\omega$ is the cardinality of $\mathbb{N}$).

Computation in the $\pi$-calculus with pairing is expressed via the exchange of *data*.

**Definition 2.3** (Reduction) *i*) The *reduction relation* over the processes of the $\pi$-calculus is defined by following (elementary) rules:

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

*ii*) As usual, we write $\to_\pi^+$ for the transitive closure of $\to_\pi$, and $\to_\pi^*$ for its reflexive and transitive closure ; we 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 $\alpha$-conversion has taken place during the synchronisation.

Part of our results will be shown using contextual equivalence, which is defined as follows:

**Definition 2.4** (Contextual equivalence) *i*) 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 $n \neq b_1 \ldots b_m$.

*ii*) We write $P\Downarrow n$ ($P$ *will output on* $n$) if there exists $Q$ such that $P \to_\pi^* Q$ and $Q\downarrow n$.

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

*iv*) We will use $\rightsquigarrow_{\mathrm{G}}$ (called *garbage collection*) when we ignore a (sub)process because it is mute, *i.e* contextually equivalent to $0$: $P \mid Q \rightsquigarrow_{\mathrm{G}} P$ if $Q \sim_{\mathrm{C}} 0$.

Notice that $\rightsquigarrow_{\mathrm{G}} \subseteq \sim_{\mathrm{C}}$; we will not use this equivalence in the opposite direction.

The following is a well-known result:

*Proposition 2.5 Assume $P$ does not contain $a$; then*

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

This expresses that synchronisation over internal channels is unobservable.

# 3 The Lambda Calculus (and variants thereof)

In this paper we will treat a number of different variants of the $\lambda$-calculus, and different notions of reduction thereon, which we will present in this section. We assume the reader to be familiar with the $\lambda$-calculus and just repeat the definition of the relevant notions. We will look in particular at Bloo and Rose's calculus $\lambda\mathbf{x}$ [13], a version of the $\lambda$-calculus with *explicit substitution*, and state our results for $\lambda\mathbf{x}$.

We will treat in the first part of this paper mainly Call-By-Name reduction systems, in the sense that, in an application $MN$, reduction will take place only in $M$ until either (1) we reach in an abstraction $\lambda x.P$, after which we can 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 *head* reduction, where we also can contract (head) redexes inside an abstraction. Since these notions are defined by limiting the *contextual* reduction rules of the $\lambda$-calculus, in all notions we present here those rules are present; this is in contrast to normal presentations that leave the contextual rules implicit. Moreover, in view of the fact that we aim to build translations of these notions of reduction into the $\pi$-calculus where the translation of normal reduction is intricate, we will consider versions of those two notions with *explicit substitution*, that can be accurately encoded.

**Definition 3.1** (Lambda terms and $\beta$-contraction [10]) *i*) The set $\Lambda$ of $\lambda$-*terms* is defined by the grammar:

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

The variable $x$ is bound in $\lambda x.M$, and free in $M$ if it occurs in $M$ but is not bound; a term is closed if it has no free variables. We write $\lambda\vec{x}.M\vec{N}$ for $\lambda x_1 \cdots x_n.MN_1 \cdots N_m$, with $n, m \geq 0$.

*ii*) The one-step reduction relation $\rightarrow_\beta$ is defined by the rules:

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

$$M \rightarrow N \ \Rightarrow \ \begin{cases} ML & \rightarrow \ NL \\ LM & \rightarrow \ LN \\ \lambda x.M & \rightarrow \ \lambda x.N \end{cases}$$

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

*iii*) The reduction relation $\rightarrow_\beta^+$ is defined as the transitive closure of $\rightarrow_\beta$, $\rightarrow_\beta^*$ is the reflexive, transitive closure of $\rightarrow_\beta$, and $=_\beta$ the smallest equivalence relation generated by $\rightarrow_\beta^*$.

*iv*) The *head-normal forms* with respect to $\beta$-reduction are defined through the grammar:

$$H \ ::= \ x\vec{M} \ (n \geq 0) \mid \lambda x.H$$

*v*) If $M = \lambda\vec{y}.(\lambda z.P)Q\vec{M}$, $(\lambda z.P)Q$ is called the *head redex* of $M$; if $M = \lambda\vec{y}.x\vec{M}$, $x$ is called the *head variable* of $M$, denoted $hv(M)$.

We adhere to Barendregt's convention, so assume that bound and free variables are always distinct, and assume that reduction is capture avoiding, using $\alpha$-conversion whenever necessary.

### 3.1 Lazy and head reduction

In implementations (and semantics) of the $\lambda$-calculus, implicit substitution $[N/x]$ on terms creates particular problems; many approaches to address this exist, varying from string reduction, $\lambda$-graphs, and Krivine's machine [23]. Krivine's machine essentially reduces $(\lambda x.M)N$ to $M\langle x := N\rangle$, where $\langle x := N\rangle$ represents a *closure*, that gets placed in an environment. The actual replacement of $N$ for $x$ will only take place if $x$ is at the start of $M$, *i.e.* is its head-variable; since $x$ might appear elsewhere, the closure cannot be removed and the environment never shrinks. Krivine's machine is deterministic and stops at weak-head normal from, *i.e.* does not reduce under an abstraction. A closure gets applied only to the head-variable, and since the environment is seen as a set of closures, the substitution rule can be formulated as

$$(x\vec{M})\,\overrightarrow{\langle y := L\rangle} \ \rightarrow \ (N\vec{M})\,\overrightarrow{\langle y := L\rangle} \quad \textit{if } \langle x := N\rangle \in \overrightarrow{\langle y := L\rangle}$$

Since closures are never removed from the environment, the latter can only increase during reduction. Notice that the normal form of a term corresponds to a term that has still (all) substitutions pending in the environment. We will see that this strategy closely corresponds to translations of $\lambda$-term reduction in the $\pi$-calculus.

To encode the contraction $(\lambda x.M)N \rightarrow_\beta M[N/x]$ in the $\pi$-calculus, implicit substitution has to be modelled using synchronisation, since this is the only computational action in the $\pi$-calculus. However, remark that synchronisation takes place one-at-the-time (*i.e. one output*

synchronising with *one input*), so each occurrence of $x$ gets serviced separately; 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 $\pi$ introduces replication, it is clear that also in the latter, the interpretation of $N$ must appear replicated in the same way.

As can be seen from the formulation of Milner's result (see Theorem 4.3), since $!P \equiv P \mid !P$, even when all $x$s 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 *input* guard (as in $!x(w).[\![N]\!]^{\mathrm{M}}_{\mathrm{I}}\, w$), making the synchronisation over $x$ the deblocking action. Since the definition of reduction on the $\pi$-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 $\lambda$-abstraction is modelled by Milner via *input*, reduction under an abstraction cannot be modelled. These two restrictions imply that, using Milner's approach which results in his encoding of the $\lambda$-calculus into the (synchronous, monadic) $\pi$-calculus as defined in Definition 4.1, only the *lazy* $\lambda$-calculus can be modelled, as defined below.

**Definition 3.2** (Lazy and head reduction) *i*) *Lazy* reduction for the $\lambda$-calculus [3] is defined by limiting the one-step reduction relation to:

$$\begin{array}{rcl} (\lambda x.M)N & \to & M[N/x] \\ M \to N & \Rightarrow & ML \to NL \end{array}$$

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

*ii*) We define *head* reduction by limiting one-step reduction to:

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

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

This notion of reduction is shown to be head-normalising (with respect to $\to_\beta$) in [11] (even quasi-head normalising); in fact, the normal forms for head reduction are head-normal forms for normal reduction [38].

Notice that head reduction encompasses lazy reduction, since $M \to_{\mathrm{L}} N$ implies $M \to_{\mathrm{H}} N$, but not vice-versa.

*Example 3.3* There is a big difference between lazy and head reduction: take $\lambda y.(\lambda x.xx)(\lambda x.xx)$, then this term is in lazy normal form, since the one redex in this term occurs under abstraction. The term clearly has no head-normal form, since $(\lambda x.xx)(\lambda x.xx)$ reduces only to itself. For a less evident example, that uses the fixedpoint constructor as used to model recursion in functional languages, take $(\lambda f.(\lambda x.f(xx))(\lambda y.f(yy)))(\lambda ab.ab)$ which reduces as follows:

$$\begin{array}{ll} (\lambda f.(\lambda x.f(xx))(\lambda y.f(yy)))(\lambda ab.ab) & \to^*_{\mathrm{L}} \\ \lambda b.((\lambda y.(\lambda ab.ab)(yy))(\lambda y.(\lambda ab.ab)(yy)))\, b & \to^*_{\mathrm{H}} \\ \lambda b.((\lambda y.(\lambda ab.ab)(yy))(\lambda y.(\lambda ab.ab)(yy)))\, b & \to^*_{\mathrm{H}} \ \cdots \end{array}$$

The first part of this reduction (in lazy reduction) terminates in the second term, which is in lazy normal form. We can then, of course, continue using head reduction; after three

steps, we reach the second term again, so this reduction does not terminate: notice that no head-variable is produced, so the term has no head normal form and is a meaningless term. This implies of course that lazy reduction is not suitable to define semantics for the $\lambda$-calculus, and only serves as a rather weak and minimal reduction strategy in the context of programming languages, fine for a reduction strategy that is only interested in reduction upto values, but not an alternative for $\beta$-reduction.

As argued by Sestoft [34], an implementation of the $\lambda$-calculus reduction must do more than represent lazy reduction, and model reductions under $\lambda$-abstractions as well. For example, taking the $\lambda$-terms

$$\begin{array}{rcl} \mathit{two} & \triangleq & \lambda fx.f(fx) \\ \mathit{four} & \triangleq & \lambda fx.f(f(f(fx))) \\ \mathit{add} & \triangleq & \lambda mnfx.mf(nfx) \end{array}$$

then it is easy to check that

$$\begin{array}{rcll} \mathit{add\ two\ two} & \triangleq & \lambda mnfx.mf(nfx)\ \mathit{two\ two} & \\ & \to^* & \lambda fx.\mathit{two}\, f(\mathit{two}\, fx) & \\ & \triangleq & \lambda fx.(\lambda gy.g(gy))f(\mathit{two}\, fx) & \\ & \to^* & \lambda fx.f(f(\mathit{two}\, fx)) & \\ & \triangleq & \lambda fx.f(f((\lambda gy.g(gy))fx)) & \\ & \to^* & \lambda fx.f(f(f(fx))) & \triangleq \mathit{four} \end{array}$$

However, lazy reduction would not go beyond the first step, since reduction under an abstraction is prohibited; thus *add two two* does not reduce to *four*. So, using Church numerals, we cannot model computability using lazy reduction.

### 3.2 Reduction with explicit substitution

It is worthwhile to note that, although not mentioned in [25], the proof of Milner's main result (Theorem 4.3 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 $x\vec{M}[N/x]$ and $N\vec{M}[N/x]$ to be *different*, whereas in the standard $\lambda$-calculus these terms are *identical*. Under explicit substitution, however, the terms $x\vec{M}\langle x{:=}N\rangle$ and $N\vec{M}\langle x{:=}N\rangle$ differ. It is therefore opportune to switch our attention to a calculus with explicit substitution; we take Bloo and Rose's calculus $\lambda\mathbf{x}$ [13], where a $\beta$-reduction of the $\lambda$-calculus is split into several more atomic steps of computation. 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 $\lambda$-calculus* $\lambda\mathbf{x}$ is an extension of that of the $\lambda$-calculus.

**Definition 3.4** (Explicit $\lambda$-calculus $\lambda\mathbf{x}$ cf. [13]) *i*) The syntax of the *explicit lambda calculus* $\lambda\mathbf{x}$ is defined by:

$$M,N \ ::= \ x \mid \lambda x.M \mid MN \mid M\langle x{:=}N\rangle$$

The notion of bound variable is extended by: $x$ us bound in $M\langle x{:=}N\rangle$.

*ii*) Let $m,n,k \geq 0$, then $hv(\lambda\vec{y}.x\vec{M}\langle z_1{:=}N_1\rangle\cdots\langle z_k{:=}N_k\rangle) = x$.

*iii* ) The reduction relation $\to_{\mathbf{x}}$ on terms in $\lambda\mathbf{x}$ is defined by the following rules:

$$
\begin{array}{rrcll}
(\beta): & (\lambda x.M)N & \to & M\langle x:=N\rangle & \\
(\mathsf{Abs}): & (\lambda y.M)\langle x:=L\rangle & \to & \lambda y.(M\langle x:=L\rangle) & \\
(\mathsf{App}): & (MN)\langle x:=L\rangle & \to & M\langle x:=L\rangle N\langle x:=L\rangle & \\
(\mathsf{Var}): & x\langle x:=L\rangle & \to & L & \\
(\mathsf{gc}): & M\langle x:=L\rangle & \to & M, & (x\notin fv(M))
\end{array}
$$

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

*iv* ) We write $\to_{:=}$ if only the rules $(\mathsf{Abs})$, $(\mathsf{App})$, $(\mathsf{Var})$, and $(\mathsf{gc})$ are applied in the reduction.

*v* ) If $M=(\lambda\vec{y}.(\lambda z.P)Q\vec{M})\overrightarrow{\langle v:=N\rangle}$, $(\lambda z.P)Q$ is called the *head redex* of $M$; if $M=(\lambda\vec{y}.x\vec{M})\overrightarrow{\langle v:=N\rangle}$, $x$ is called the *head variable* of $M$, denoted $hv(M)$.

Since the main rules have no critical pair, reduction is confluent.

Notice that these rules allow reduction to take place also inside the substitution term, as expressed by the last rule.

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

*Proposition 3.5* ($\lambda\mathbf{x}$ IMPLEMENTS $\beta$-REDUCTION) *i* ) $M\to_\beta N \Rightarrow M\to_{\mathbf{x}}^* N$.
*ii* ) $M\in\Lambda$ & $M\to_{\mathbf{x}}^* N \Rightarrow \exists L\in\Lambda\ [N\to_{:=}^* L$ & $M\to_\beta^* L]$.

We will show our main results for variants of $\lambda\mathbf{x}$; since $\lambda\mathbf{x}$ implements $\beta$-reduction, we also show our results for normal $\beta$-reduction (with implicit substitution).

In [9], two restrictions of reduction in $\lambda\mathbf{x}$ were defined which are useful for the results of this paper. The main idea was to formally define a version of $\lambda\mathbf{x}$ that allows for, as for Krivine's machine, only the head-variable of a term to be replaced. We deviate 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. 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, *i.e.* after the redex $(\lambda y.y)x$ gets contracted, yielding $x\langle x:=\lambda y.y\rangle$; now the variable is at the head, the postponed substitution can be applied which in turn yields $\lambda y.y$. So, in general, lazy explicit substitution replaces only the lazy head variable of a term.

### 3.3 Explicit lazy and head reduction

We will distinguish *two* notions of explicit reduction, namely explicit *lazy* reduction and explicit *head* reduction; we will show that the first is modelled by Milner's translation, and the second by our translation, up to renaming.

**Definition 3.6** (Explicit head reduction cf. [9]) We define *explicit head reduction* $\to_{\mathbf{xH}}$ on $\lambda\mathbf{x}$ in the spirit of $\to_{\mathbf{x}}$, but give a central role to the notion of head-variable.

*i* ) The reduction rules are:

$$
\begin{array}{rrcll}
(\beta): & (\lambda x.M)N & \to & M\langle x:=N\rangle & \\
(\mathsf{Abs}): & (\lambda y.M)\langle x:=N\rangle & \to & \lambda y.(M\langle x:=N\rangle) & \\
(\mathsf{App}): & y\overrightarrow{M}\overrightarrow{\langle x:=L\rangle}\langle y:=N\rangle & \to & N\overrightarrow{M}\overrightarrow{\langle x:=L\rangle}\langle y:=N\rangle & \\
(\mathsf{gc}): & M\langle x:=L\rangle & \to & M, & (x\notin fv(M))
\end{array}
$$

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

*ii* ) We define *explicit lazy reduction* $\to_{\mathbf{xL}}$ by eliminating rule $(\mathsf{Abs})$, as well as the contextual rule

$$M\to N \;\Rightarrow\; \lambda x.M\to\lambda x.N$$

*iii* ) We define $\to_{\mathbf{xH}}^{\not{c}}$ and $\to_{\mathbf{xL}}^{\not{c}}$ as variants of $\to_{\mathbf{xH}}$ and $\to_{\mathbf{xL}}$, respectively, that omit rule $(\mathsf{gc})$.

Notice that the $(\mathsf{App})$ reduction is the one implicitly used by Milner [25] in Lemma 4.5 (case 3), and corresponds to reduction in Krivine's machine.

In both the notions of Definition 3.6, substitution is lazy: notice that we are reducing only at the head of a term, and only when a variable mentioned in the pending substitutions appears in the head will that substitution be executed; this appears to be the implicit approach of [25] (see Lemma 4.5, case 3).

Explicit lazy reduction of $\lambda\mathbf{xL}$ has similarities with Krivine's machine [23], 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 head reduction.

*Example 3.7* *i* ) In $\to_{\mathbf{xH}}$ we can reduce as follows:

$$
\begin{array}{lll}
(\lambda x.xx)(\lambda y.y) & \to_{\mathbf{xH}}(\beta) & xx\langle x:=\lambda y.y\rangle \\
 & \to_{\mathbf{xH}}(\mathsf{App}) & (\lambda z.z)x\langle x:=\lambda y.y\rangle \\
 & \to_{\mathbf{xH}}(\beta) & z\langle z:=x\rangle\langle x:=\lambda y.y\rangle \\
 & \to_{\mathbf{xH}}(\mathsf{App}) & x\langle z:=x\rangle\langle x:=\lambda y.y\rangle \\
 & \to_{\mathbf{xH}}(\mathsf{gc}) & x\langle x:=\lambda y.y\rangle \\
 & \to_{\mathbf{xH}}(\mathsf{App}) & (\lambda z.z)\langle x:=\lambda y.y\rangle \qquad \to_{\mathbf{xH}}(\mathsf{gc})\;\; \lambda z.z
\end{array}
$$

Alternatively, we can execute $(\mathsf{gc})$ at different moments, but the end result will be the same. Notice the use of $\alpha$-conversion.

*ii* ) Of course in $\Lambda\mathbf{xH}$ we can have non-terminating reductions, as illustrated by the following reduction:

$$
\begin{array}{rcl}
(\lambda x.xx)(\lambda x.xx) & \to_{\mathbf{x}\mathrm{H}} & (\beta) \\
xx\,\langle x := \lambda x.xx\rangle & \to_{\mathbf{x}\mathrm{H}} & (\mathsf{App}), (\alpha) \\
(\lambda y.yy)x\,\langle x := \lambda y.yy\rangle & \to_{\mathbf{x}\mathrm{H}} & (\beta) \\
yy\,\langle y := x\rangle\,\langle x := \lambda y.yy\rangle & \to^*_{\mathbf{x}\mathrm{H}} & (\mathsf{App}) \\
xy\,\langle y := x\rangle\,\langle x := \lambda y.yy\rangle & \to^*_{\mathbf{x}\mathrm{H}} & (\mathsf{App}), (\alpha) \\
(\lambda z.zz)y\,\langle y := x\rangle\,\langle x := \lambda y.yy\rangle & \to_{\mathbf{x}\mathrm{H}} & (\beta) \\
zz\,\langle z := y\rangle\,\langle y := x\rangle\,\langle x := \lambda y.yy\rangle & \to_{\mathbf{x}\mathrm{H}} & \cdots
\end{array}
$$

(notice the $\alpha$-conversions, needed to adhere to Barendregt's convention). This reduction is deterministic and clearly does not terminate; notice that $(\lambda x.xx)(\lambda x.xx)$ does not run to itself; moreover, all terms are themselves $\to_{\mathbf{x}\mathrm{H}}$-redexes. However:

$$
\begin{array}{rcl}
zz\,\langle z := y\rangle\,\langle y := x\rangle\,\langle x := \lambda y.yy\rangle & \to^*_{:=} & \\
yy\,\langle y := x\rangle\,\langle x := \lambda y.yy\rangle & \to^*_{:=} & \\
xx\,\langle x := \lambda y.yy\rangle & \to^*_{:=} & (\lambda y.yy)(\lambda y.yy)
\end{array}
$$

The following proposition states the relation between explicit head reduction, head reduction, and explicit head reduction. Using Proposition 3.5, we can easily show:

*Proposition 3.8* *i ) If $M \to_\beta N$ by contracting a head redex, then there exists $P$ such that $M \to^*_{\mathbf{x}\mathrm{H}} P$ and $P \to^*_{:=} N$.*

*ii ) If $M \to^*_{\mathrm{H}} N$, and $N$ is in $\to_{\mathrm{H}}$-normal form* (i.e. *in $\beta$-head-normal form), then there exists $L \in \lambda\mathbf{x}$ such that $L$ is in $\Lambda\mathbf{x}\mathrm{H}$-normal form, and $M \to^*_{\mathbf{x}\mathrm{H}} L$ and $L \to^*_{:=} N$.*

*iii ) If $M \to^*_{\mathrm{H}} N$, then there exists $L$ such that $M \to^*_{\mathbf{x}\mathrm{H}} L$ and $L \to^*_{:=} N$.*

*iv ) If $M \to^*_{\mathrm{H}} N$, then there exists $P, \vec{x}, \vec{Q}$ such that $M \to^*_{\mathbf{x}\mathrm{H}} P\langle\overrightarrow{x = Q}\rangle \to^*_{:=} N$, and $P[\overrightarrow{Q/x}] = N$.*

*v ) If $M \to^*_{\mathrm{L}} N$, then there exists $L \in \lambda\mathbf{x}$ such that $M \to^*_{\mathbf{x}\mathrm{L}} L$ and $L \to^*_{:=} N$.*

Notice that, in particular, the first part holds for *single step* reductions. Since head reduction reduces a term $M$ to $\beta$-head-normal form, if it exists, this implies that also $\to_{\mathbf{x}\mathrm{H}}$ reduces to head-normal form, albeit with perhaps some substitutions still pending.

*Proposition 3.9* *i ) If $M \downarrow_{\mathbf{x}\mathrm{H}} N$, then there exists a pure $\lambda$-term $L$ such that $N \to_{:=} L$ and $M \to^*_\beta L$, and $L$ is in $\Lambda$ head-normal form.*

*ii ) If $M \downarrow_{\mathbf{x}\mathrm{L}} N$, then there exists a pure $\lambda$-term $L$ such that $N \to_{:=} L$ and $M \to^*_\beta L$, and $L$ is in $\Lambda$ head-normal form.*

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

**Definition 3.10** (CF. [24]) The normal forms with respect to $\to_{\mathbf{x}\mathrm{H}}$ are defined through the grammar:

$$\mathbf{N} \;::=\; x\overrightarrow{M}\ (n \geq 0) \;\mid\; \lambda x.\mathbf{N} \;\mid\; \mathbf{N}\langle x := M\rangle \quad (hv(\mathbf{N}) \neq x)$$

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

Notice that the substitution applied to an application $MN\,\langle x := L\rangle$ remains on the outside, and will only disappear when the result of running that term will generate a term that no longer has occurrences of $x$; this is not a feature of Krivine's machine.

*Example 3.11* *i*) The rule (App) is necessary in order to be able to run:

$$\begin{array}{rcl}(\lambda x.(\lambda y.xN)M)L & \to_{\mathrm{xH}} & (\lambda y.xN)M\,\langle x{:=}L\rangle \\ & \to_{\mathrm{xH}} & xN\,\langle y{:=}M\rangle\,\langle x{:=}L\rangle \\ & \to_{\mathrm{xH}} & LN\,\langle y{:=}M\rangle\,\langle x{:=}L\rangle\end{array}$$

*ii*) Substitutions are left after reducing, like $(\lambda z.yz)N \to_{\mathrm{xH}} yz\,\langle z{:=}N\rangle$; the latter term is in $\to_{\mathrm{xH}}$-normal form, since $z$ is not the head variable.

# 4 Milner's encoding results

Translations of the $\lambda$-calculus into the $\pi$-calculus have been studied in detail over the last two decades. The first and most prominent one – in the sense that almost all later translations are variants – was defined by Milner [25]; that translation provides an abstract machine, in spirit very similar to Krivine's abstract machine. Notice that the rule (gc) is not modelled by Krivine's machine, since the set of closures is only ever extended, never made smaller.

## 4.1 Milner's encoding

In his seminal paper [25], Milner defines a translation of the $\lambda$-calculus into the (synchronous, monadic) $\pi$-calculus, and shows some correctness results. His Call-By-Name translation is inspired by the normal semantics of $\lambda$-terms, which states for abstraction:

$$\llbracket \lambda x.M \rrbracket^{\mathcal{M}}_{\xi} \;=\; G\,(\boldsymbol{\lambda}\, d \in \mathcal{M}.\llbracket M \rrbracket^{\mathcal{M}}_{\xi(d/x)})$$

So, also in the translation, instead of executing $M[N/x]$, the interpretation of $M$ is executed in an environment that binds $N$ to the variable $x$. This leads to:

**Definition 4.1** (Milner's interpretation [25]) *Input-based translation* of $\lambda$-terms into the synchronous $\pi$-calculus is defined by:

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

Milner calls $\llbracket x := M \rrbracket^{\mathrm{M}}$ an "*environment entry*"; it could be omitted from the definition above, but is of use separately. In $\llbracket \cdot \rrbracket^{\mathrm{M}} a$, the name $a$ is the channel along which $\llbracket M \rrbracket^{\mathrm{M}}$ receives its argument; this is used to communicate with the interpretation of the argument, as made clear in the third case, where 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.

Notice that both the body of the abstraction and the argument in an application get positioned under an input, and that therefore reductions inside these subterms cannot be modelled; as a result, the simulation via the translation is limited to lazy reduction. It is possible to improve on this result by extending the notion of reduction or congruence on $\pi$, by adding, as done in [22], $P \to_\pi Q \Rightarrow x(v).\,P \to_\pi x(v).\,Q$, but we will not follow that approach here.

*Example 4.2* Using $\llbracket \cdot \rrbracket^{\mathrm{M}} \cdot$, the interpretation of a $\beta$-redex (only) reduces as follows:

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

$$
\begin{array}{lll}
 & \llbracket (\lambda x.xx)(\lambda y.y) \rrbracket^{\mathbf{M}} a & \stackrel{\Delta}{=} \\
1: & (\nu c)\,(\llbracket \lambda x.xx \rrbracket^{\mathbf{M}} c \mid (\nu z)\,(\overline{c}\langle z\rangle.\,\overline{c}\langle a\rangle.\,\llbracket z := \lambda y.y \rrbracket^{\mathbf{M}})) & \stackrel{\Delta}{=} \\
 & (\nu c)\,(c(x).\,c(b).\,\llbracket xx \rrbracket^{\mathbf{M}} b \mid (\nu z)\,(\overline{c}\langle z\rangle.\,\overline{c}\langle a\rangle.\,\llbracket z := \lambda y.y \rrbracket^{\mathbf{M}})) & \to_{\pi}^{+} (c) \\
2: & (\nu z)\,(\llbracket zz \rrbracket^{\mathbf{M}} a \mid \llbracket z := \lambda y.y \rrbracket^{\mathbf{M}}) & \stackrel{\Delta}{=} \\
 & (\nu z)\,((\nu c)\,(\overline{z}\langle c\rangle \mid (\nu z_1)\,(\overline{c}\langle z_1\rangle.\,\overline{c}\langle a\rangle.\,\llbracket z_1 := z \rrbracket^{\mathbf{M}})) \mid !z(w).\,\llbracket \lambda y.y \rrbracket^{\mathbf{M}} w) & \to (z) \\
3: & (\nu z)\,((\nu c)\,(\llbracket \lambda y.y \rrbracket^{\mathbf{M}} c \mid (\nu z_1)\,(\overline{c}\langle z_1\rangle.\,\overline{c}\langle a\rangle.\,\llbracket z_1 := z \rrbracket^{\mathbf{M}})) \mid \llbracket z := \lambda y.y \rrbracket^{\mathbf{M}}) & \stackrel{\Delta}{=} \\
 & (\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.\,\llbracket z_1 := z \rrbracket^{\mathbf{M}})) \mid \llbracket z := \lambda y.y \rrbracket^{\mathbf{M}}) & \to_{\pi}^{+} (c) \\
4: & (\nu z)\,((\nu z_1)\,(\overline{z_1}\langle a\rangle \mid !z_1(w).\,\overline{z}\langle w\rangle) \mid \llbracket z := \lambda y.y \rrbracket^{\mathbf{M}}) & \to (z_1) \\
5: & (\nu z)\,((\nu z_1)\,(\overline{z}\langle a\rangle \mid \llbracket z_1 := z \rrbracket^{\mathbf{M}}) \mid \llbracket z := \lambda y.y \rrbracket^{\mathbf{M}}) & \rightsquigarrow_{\mathrm{G}} \\
6: & (\nu z)\,(\overline{z}\langle a\rangle \mid !z(w).\,\llbracket \lambda y.y \rrbracket^{\mathbf{M}} w) & \to_{\pi} (z) \\
7: & (\nu z)\,(\llbracket \lambda y.y \rrbracket^{\mathbf{M}} a \mid \llbracket z := \lambda y.y \rrbracket^{\mathbf{M}}) & \rightsquigarrow_{\mathrm{G}} \\
8: & \llbracket \lambda y.y \rrbracket^{\mathbf{M}} a &
\end{array}
$$

Figure 1: Running Milner's translation of $(\lambda x.xx)(\lambda y.y)$ of Example 4.4.

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 $\llbracket M \rrbracket^{\mathbf{M}} a$ reaches the point where output is generated over $x$. This implies of course that we can model reductions in $M$ that take place *before* the substitution gets executed, *i.e.* 'under the abstraction', but *after* a first step in the evaluation of the redex: this implies that Milner's encoding represents *more* than just lay reduction; see Theorem 4.6.

Notice the 'delay' mechanism that forms part of Milner's translation through the guard in $!z(w).\,\llbracket N \rrbracket^{\mathbf{M}} w$. This has not only the advantage that reduction cannot take place 'in the argument of substitution' $\llbracket N \rrbracket^{\mathbf{M}} w$; also, no context can interact with this process, even after a copy has been peeled off (via $!x(w).\,\llbracket N \rrbracket^{\mathbf{M}} w \equiv x(w).\,\llbracket N \rrbracket^{\mathbf{M}} w \mid \llbracket x := N \rrbracket^{\mathbf{M}}$) the process $\llbracket N \rrbracket^{\mathbf{M}} x$ cannot start running, since guarded by an input.

Milner states the main correctness result for his interpretation with respect to lazy reduction as follows:

**Theorem 4.3** ([25]) *For all closed $\lambda$-terms $M$, either:*
*i ) $M \to_{\mathrm{L}} \lambda y.R[\overline{N/x}]$, and $\llbracket M \rrbracket^{\mathbf{M}} u \to_{\pi}^{*} (\overline{\nu x})\,(\llbracket \lambda y.R \rrbracket^{\mathbf{M}} u \mid \overline{\llbracket x := N \rrbracket^{\mathbf{M}}})$, or*
*ii ) both $M$ and $\llbracket M \rrbracket^{\mathbf{M}} u$ diverge.*

It is worthwhile to note that, although not mentioned in [25], in the proof of this result Milner treats the substitution as *explicit*, not as *implicit*; for example, in the proof of Lemma 4.5 in that paper, case 3 treats the term $x\overrightarrow{M}[N/x]$ and $N\overrightarrow{M}[N/x]$ as *different*. In fact, although stated with implicit substitution, Milner's result does not show that lazy reduction (with implicit substitution) is fully modelled, as made clear in the next example.

*Example 4.4* In Figure 1, we show how to run the term $(\lambda x.xx)(\lambda y.y)$ under Milner's interpretation. Notice that there we executed the only possible synchronisations, and that in the reduction path no term corresponds to

$$(\nu c)\,(\llbracket \lambda y.y \rrbracket^{\mathbf{M}} c \mid (\nu z)\,(\overline{c}\langle z\rangle.\,\overline{c}\langle a\rangle.\,\llbracket z := \lambda y.y \rrbracket^{\mathbf{M}}))$$

(*i.e.* $\llbracket (\lambda y.y)(\lambda y.y) \rrbracket^{\mathbf{M}} a$). So, in particular, $(\nu z)\,(\llbracket zz \rrbracket^{\mathbf{M}} a \mid \llbracket z := \lambda y.y \rrbracket^{\mathbf{M}})$ does *not* reduce to the result of the substitution $\llbracket (\lambda y.y)(\lambda y.y) \rrbracket^{\mathbf{M}} a$. Of course reducing the term $\llbracket (\lambda y.y)(\lambda y.y) \rrbracket^{\mathbf{M}} a$ will also yield $\llbracket \lambda y.y \rrbracket^{\mathbf{M}} a$, but we can only show that the two processes $\llbracket (\lambda x.xx)(\lambda y.y) \rrbracket^{\mathbf{M}} a$ and $\llbracket (\lambda y.y)(\lambda y.y) \rrbracket^{\mathbf{M}} a$ have a *common reduct*, not that the first reduces to the second. So it is

impossible to reduce the translation of $(\lambda x.xx)(\lambda y.y)$ to that of $(\lambda y.y)(\lambda y.y)$, and, thereby Milner's translation does not model implicit substitution itself.

### 4.2 Milner's encoding and explicit lazy reduction

Although intuitively $(\nu x)\,(\llbracket M \rrbracket^{\mathrm{M}} a \mid \llbracket x := N \rrbracket^{\mathrm{M}})$ can be seen as $\llbracket M[N/x] \rrbracket^{\mathrm{M}} a$, Example 4.4 illustrates that these processes are substantially different. So is there then perhaps a relation with *explicit* substitution? By close inspection it becomes clear that the reduction in Figure 1 actually simulates:

$$\begin{array}{rcll}
(\lambda x.xx)(\lambda y.y) & \to & (zz)\,\langle z := \lambda y.y\rangle & \\
 & \to & ((\lambda y.y)z)\,\langle z := \lambda y.y\rangle & \\
 & \to & z_1\,\langle z_1 := z\rangle\,\langle z := \lambda y.y\rangle & \\
 & \to & z\,\langle z_1 := z\rangle\,\langle z := \lambda y.y\rangle & \\
 & \to & (\lambda y.y)\,\langle z_1 := z\rangle\,\langle z := \lambda y.y\rangle & \\
 & \to & (\lambda y.y)\,\langle z := \lambda y.y\rangle & \to\ \lambda y.y
\end{array}$$

(where the numbered lines in Figure 1 correspond to the respective terms in this reduction). Notice that this contains some unnecessary $\alpha$-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 a reduction in $\to_{\mathbf{xL}}$. It is clear that $(\nu x)\,(\llbracket M \rrbracket^{\mathrm{M}} a \mid \llbracket x := N \rrbracket^{\mathrm{M}})$ is more related to the explicit substitution, a fact we will exploit in Definition 4.5.

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 4.5** We extend the interpretation of Definition 4.1 to $\lambda\mathbf{x}$ (and $\lambda\mathbf{xL}$ and $\Lambda\mathbf{xH}$) by adding the case:

$$\llbracket M\langle x := N\rangle \rrbracket^{\mathrm{M}} a \ \stackrel{\Delta}{=}\ (\nu x)\,(\llbracket M \rrbracket^{\mathrm{M}} a \mid \llbracket x := N \rrbracket^{\mathrm{M}})$$

We can now show the that single explicit lazy reduction steps are preserved by Milner's interpretation:

**Theorem 4.6** ($\llbracket \cdot \rrbracket^{\mathrm{M}} \cdot$ PRESERVES $\to_{\mathbf{xL}}$) *If $M \to_{\mathbf{xL}} N$, then $\llbracket M \rrbracket^{\mathrm{M}} a \to_{\pi}^{*}, \rightsquigarrow_{\mathrm{G}}^{*} \llbracket N \rrbracket^{\mathrm{M}} a$.*

*Proof:* By induction on the definition of explicit lazy reduction; we only show the interesting base cases.

$$\begin{array}{lll}
((\lambda y.M)N \to_{\mathbf{xL}} M\,\langle y := N\rangle): & \llbracket (\lambda y.M)N \rrbracket^{\mathrm{M}} a & \to_{\pi}^{+}\ (4.2) \\
\quad (\nu y)\,(\llbracket M \rrbracket^{\mathrm{M}} a \mid \llbracket y := N \rrbracket^{\mathrm{M}}) & & \stackrel{\Delta}{=}\ \llbracket M\langle y := N\rangle \rrbracket^{\mathrm{M}} a \\
(y\overrightarrow{M}\overrightarrow{\langle x := L\rangle}\,\langle y := N\rangle \to N\overrightarrow{M}\overrightarrow{\langle x := L\rangle}\,\langle y := N\rangle): & \llbracket y\overrightarrow{M}\overrightarrow{\langle x := L\rangle}\,\langle y := N\rangle \rrbracket^{\mathrm{M}} a & \stackrel{\Delta}{=} \\
\quad (\nu y\vec{x})\,(\llbracket y\overrightarrow{M} \rrbracket^{\mathrm{M}} a \mid \overline{\llbracket x := L \rrbracket^{\mathrm{M}}} \mid \llbracket y := N \rrbracket^{\mathrm{M}}) & & \stackrel{\Delta}{=} \\
\quad (\nu y\vec{x})\,((\nu c_n)\,(\llbracket yM_1 \cdots M_{n-1} \rrbracket^{\mathrm{M}} c_n \mid (\nu z)\,(\overline{c_n}\langle z\rangle.\,\overline{c_n}\langle a\rangle.\,\llbracket z := M_n \rrbracket^{\mathrm{M}})) \mid \overline{\llbracket x := L \rrbracket^{\mathrm{M}}} \mid \llbracket y := N \rrbracket^{\mathrm{M}}) & & \stackrel{\Delta}{=}, \equiv \\
\quad (\nu y\vec{x})\,((\nu c_n \cdots c_1)\,(\overline{y}\langle c_1\rangle \mid (\nu z)\,(\overline{c_1}\langle z\rangle.\,\overline{c_1}\langle a\rangle.\,\llbracket z := M_1 \rrbracket^{\mathrm{M}}) \mid \cdots & & \\
\qquad\qquad (\nu z)\,(\overline{c_n}\langle z\rangle.\,\overline{c_n}\langle a\rangle.\,\llbracket z := M_n \rrbracket^{\mathrm{M}})) \mid \overline{\llbracket x := L \rrbracket^{\mathrm{M}}} \mid !\,y(w).\,\llbracket N \rrbracket^{\mathrm{M}} w \mid \llbracket y := N \rrbracket^{\mathrm{M}}) & & \to_{\pi}(y) \\
\quad (\nu y\vec{x})\,((\nu c_n \cdots c_1)\,(\llbracket N \rrbracket^{\mathrm{M}} c_1 \mid (\nu z)\,(\overline{c_1}\langle z\rangle.\,\overline{c_1}\langle a\rangle.\,\llbracket z := M_1 \rrbracket^{\mathrm{M}}) \mid \cdots & & \\
\qquad\qquad (\nu z)\,(\overline{c_n}\langle z\rangle.\,\overline{c_n}\langle a\rangle.\,\llbracket z := M_n \rrbracket^{\mathrm{M}})) \mid \overline{\llbracket x := L \rrbracket^{\mathrm{M}}} \mid \llbracket y := N \rrbracket^{\mathrm{M}}) & & \stackrel{\Delta}{=} \\
\quad \llbracket N\overrightarrow{M}\overrightarrow{\langle x := L\rangle}\,\langle y := N\rangle \rrbracket^{\mathrm{M}} a & & \\
(M\,\langle x := P\rangle \to M,\ x \notin fv\,(M)): & \llbracket M\langle x := P\rangle \rrbracket^{\mathrm{M}} a & \stackrel{\Delta}{=} \\
\quad (\nu x)\,(\llbracket M \rrbracket^{\mathrm{M}} a \mid \llbracket x := P \rrbracket^{\mathrm{M}}) & & \equiv\ (x \notin fv\,(M)) \\
\quad \llbracket M \rrbracket^{\mathrm{M}} a \mid (\nu x)\,(!\,x(w).\,\llbracket P \rrbracket^{\mathrm{M}} w) & & \rightsquigarrow_{\mathrm{G}}\ \llbracket M \rrbracket^{\mathrm{M}} a
\end{array}$$

Notice that, in particular, we do not need the $\lambda$-terms involved to be closed, so our result is a generalisation of Milner's. Also, garbage collection is only used in the last case. Of course we can show:

*Corollary 4.7 If $M$ is closed, and $M \to^*_{\mathbf{XL}} (\lambda y.N)\overrightarrow{\langle x := L\rangle}$, then $[\![M]\!]^{\mathbf{M}} a \to^*_{\pi}, \rightsquigarrow^*_{\mathrm{G}} (\overrightarrow{\nu x})\,([\![\lambda y.N]\!]^{\mathbf{M}} a \mid \overrightarrow{[\![x := L]\!]^{\mathbf{M}}})$.*

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

Milner does not consider garbage collection for his result; since above garbage collection is only used to model rule (gc), we can restate Milner's result using $\to^{\not{\mathsf{gc}}}_{\mathbf{XL}}$.

*Corollary 4.8 If $M$ is closed, and $M \to^{\not{\mathsf{gc}}*}_{\mathbf{XL}} (\lambda y.N)\overrightarrow{\langle x := L\rangle}$, then $[\![M]\!]^{\mathbf{M}} a \to^+_{\pi} (\overrightarrow{\nu x})\,([\![\lambda y.N]\!]^{\mathbf{M}} a \mid \overrightarrow{[\![x := L]\!]^{\mathbf{M}}})$.*

Notice that, without garbage collection, $(\lambda x.xx)(\lambda y.y) \to^{\not{\mathsf{gc}}*}_{\mathbf{XL}} (\lambda y.y)\,\langle z_1 := z\rangle\,\langle z := \lambda y.y\rangle$ (see Example 4.4) where the latter is in normal form.

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

**Theorem 4.9** (Simulation of lazy reduction under Milner's encoding) *i ) If $M \to^*_{\mathrm{L}} N$, then there exists $M'$ such that $[\![M]\!]^{\mathbf{M}} a \to^+_{\pi} [\![M']\!]^{\mathbf{M}} a$, and $M' \to^*_{:=} N$.*

*ii ) If $M$ is closed and lazy normalising, then there exists $N, N', \vec{x}, \vec{L}$ such that $M \to_{\mathrm{L}} \lambda y.N$, and $[\![M]\!]^{\mathbf{M}} a \to^*_{\pi}, \rightsquigarrow^*_{\mathrm{G}} (\overrightarrow{\nu x})\,([\![\lambda y.N']\!]^{\mathbf{M}} a \mid \overrightarrow{[\![x := L]\!]^{\mathbf{M}}})$, and $\lambda y.N'\overrightarrow{\langle x := L\rangle} \to^*_{:=} \lambda x.N$.*

*Proof:* By Theorem 4.6 and Proposition 3.8. □

In Section 5.4 we will show that we can even state the last result without $\rightsquigarrow_{\mathrm{G}}$.

# 5 A logical, output-based translation of $\lambda$-terms

We have repeated above that Milner's interpretation respects lazy reduction. In this section, we will show that it is possible to deviate from Milner's original approach to the translation problem and actually make a gain in the process. Inspired by the relation between natural deduction and the sequent calculus [18], interpreting terms under *output* rather than under *input*, and using the $\pi$-calculus with pairing, we can define a different translation of the $\lambda$-calculus into the $\pi$-calculus. Although the main objective of our translation is to show the preservation of type assignment, we also achieve a more expressive translation that preserves not just lazy reduction, but also the larger notion of head reduction.

Our translation follows from – but is an improvement of – the concatenation of the translation of the $\lambda$-calculus into $\mathcal{X}$ (which established a link between natural deduction and the sequent calculus) as defined in [8], and the interpretation of $\mathcal{X}$ into the $\pi$-calculus as defined in [5]. The idea behind the translation stems from the observation that, in the $\lambda$-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*. Translating into the (multi-output) $\pi$-calculus, this locality property no longer holds; we need to specify the destination of a term, by giving a name to its output: this is what the translation does.

Our translation is defined by:

**Definition 5.1** (Output-based interpretation of $\lambda$-terms in $\pi$) The mapping $[\![\cdot]\!]^{\mathbf{H}}\cdot$ is de-

$$
\begin{array}{ll}
\llbracket (\lambda x.xx)(\lambda y.y) \rrbracket^{\mathbf{H}}_{\rrbracket} a & \triangleq \\
(\nu c)\,((\nu xb)\,(\llbracket xx \rrbracket^{\mathbf{H}}_{\rrbracket} b \mid \overline{c}\langle x,b\rangle) \mid c(v,d).\,(!\llbracket \lambda y.y \rrbracket^{\mathbf{H}}_{\rrbracket} v \mid d \twoheadrightarrow a)) & \rightarrow_{\pi} (c) \\
(\nu xb)\,(\llbracket xx \rrbracket^{\mathbf{H}}_{\rrbracket} b \mid !\llbracket \lambda y.y \rrbracket^{\mathbf{H}}_{\rrbracket} x \mid b \twoheadrightarrow a) & \triangleq \\
(\nu xb)\,((\nu c)\,(x(w).\overline{c}\langle w\rangle \mid c(v,d).\,(!\llbracket x \rrbracket^{\mathbf{H}}_{\rrbracket} v \mid d \twoheadrightarrow b)) \mid !(\nu yb)\,(\llbracket y \rrbracket^{\mathbf{H}}_{\rrbracket} b \mid \overline{x}\langle y,b\rangle) \mid b \twoheadrightarrow a) & \rightarrow_{\pi} (x) \\
(\nu xb)\,((\nu c)\,((\nu yb')\,(\llbracket y \rrbracket^{\mathbf{H}}_{\rrbracket} b' \mid \overline{c}\langle y,b'\rangle) \mid c(v,d).\,(!\llbracket x \rrbracket^{\mathbf{H}}_{\rrbracket} v \mid d \twoheadrightarrow b)) \mid !\llbracket \lambda y.y \rrbracket^{\mathbf{H}}_{\rrbracket} x \mid b \twoheadrightarrow a) & \rightarrow_{\pi} (c), \triangleq \\
(\nu xb)\,((\nu yb')\,(y(w).\overline{b'}\langle w\rangle \mid !x(w).\overline{y}\langle w\rangle \mid b' \twoheadrightarrow b) \mid !(\nu yb)\,(\llbracket y \rrbracket^{\mathbf{H}}_{\rrbracket} b \mid \overline{x}\langle y,b\rangle) \mid b \twoheadrightarrow a) & \rightarrow_{\pi} (x,y), \triangleq =_{\alpha} (z/y) \\
(\nu xb)\,((\nu yb')\,((\nu zb)\,(\llbracket z \rrbracket^{\mathbf{H}}_{\rrbracket} b \mid \overline{b'}\langle z,b\rangle) \mid !x(w).\overline{y}\langle w\rangle \mid b' \twoheadrightarrow b) \mid !\llbracket \lambda y.y \rrbracket^{\mathbf{H}}_{\rrbracket} x \mid b \twoheadrightarrow a) & \rightarrow_{\pi} (b'b), \triangleq \\
(\nu x)\,((\nu y)\,((\nu zb)\,(\llbracket z \rrbracket^{\mathbf{H}}_{\rrbracket} b \mid \overline{a}\langle z,b\rangle) \mid !x(w).\overline{y}\langle w\rangle) \mid !\llbracket \lambda y.y \rrbracket^{\mathbf{H}}_{\rrbracket} x) & \equiv \\
(\nu zb_2)\,(\llbracket z \rrbracket^{\mathbf{H}}_{\rrbracket} b_2 \mid \overline{a}\langle z,b_2\rangle) \mid (\nu xy)\,(!\llbracket x \rrbracket^{\mathbf{H}}_{\rrbracket} y \mid !\llbracket \lambda y.vy \rrbracket^{\mathbf{H}}_{\rrbracket} x) & \leadsto_{\mathrm{G}} \\
(\nu zb)\,(\llbracket z \rrbracket^{\mathbf{H}}_{\rrbracket} b \mid \overline{a}\langle z,b\rangle) & \triangleq \llbracket \lambda z.z \rrbracket^{\mathbf{H}}_{\rrbracket} a
\end{array}
$$

Figure 2: Running $\llbracket (\lambda x.xx)(\lambda y.y) \rrbracket^{\mathbf{H}}_{\rrbracket} a$.

fined by:

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

In particular:

- We see a variable $x$ as an input channel, and send its input to the output channel $a$ that we interpret it under;
- 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;
- For an application $MN$, the output of $M$, transmitted over $c$, is received as a pair $\langle v,d\rangle$ of input-output names in the right-hand side; the received input $v$ name is used as output name for the interpretation of $N$, 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 implements, as for Milner's encoding, the (distributive) substitution on $\lambda$-terms. Also, every $\llbracket N \rrbracket^{\mathbf{H}}_{\rrbracket} a$ is a process that outputs 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 observation immediately gives:

*Proposition 5.2* $(\nu a)\,(\llbracket M \rrbracket^{\mathbf{H}}_{\rrbracket} a) \sim_{\mathrm{C}} 0$.

We will use this proposition for garbage collection during reduction of interpreted terms.

*Example 5.3* To illustrate the encoding, Figure 2 shows how to reduce $\llbracket (\lambda x.xx)(\lambda y.y) \rrbracket^{\mathbf{H}}_{\rrbracket} a$ (see Example 3.7).

To underline the significance of our results, notice that the translation is not trivial, since $\lambda yz.y$ and $\lambda x.x$ are interpreted by $(\nu yb)\,((\nu zb_1)\,(y(w).\overline{b_1}\langle w\rangle \mid \overline{b}\langle z,b_1\rangle) \mid \overline{a}\langle y,b\rangle)$ and $(\nu xb)\,(x(w).\overline{b}\langle w\rangle \mid \overline{a}\langle x,b\rangle)$, respectively, processes that differ under $\sim_{\mathrm{C}}$.

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

*Example 5.4* Assume that $\llbracket \lambda x.M \rrbracket^{\mathrm{H}} a = (\nu xb)\,(\overline{a}\langle x,b\rangle.\llbracket M \rrbracket^{\mathrm{H}} b)$, then $\llbracket (\lambda y.y)(\lambda x.x) \rrbracket^{\mathrm{H}} a$ would run as follows:

$$\begin{array}{rcl}
\llbracket (\lambda y.y)(\lambda x.x) \rrbracket^{\mathrm{H}} a & \stackrel{\Delta}{=} & (\nu c)\,((\nu yb)\,(\overline{c}\langle y,b\rangle.\llbracket y \rrbracket^{\mathrm{H}} b) \mid c(v,d).\,(!\llbracket \lambda x.x \rrbracket^{\mathrm{H}} v \mid d \rightarrow a)) \\
 & \rightarrow_{\pi}(c) & (\nu yb)\,(\llbracket y \rrbracket^{\mathrm{H}} b \mid !\llbracket \lambda x.x \rrbracket^{\mathrm{H}} y \mid b \rightarrow a) \\
 & \equiv, \stackrel{\Delta}{=} & (\nu yb)\,(y(w).\,\overline{b}\langle w\rangle \mid (\nu xe)\,(\overline{y}\langle x,e\rangle.\llbracket x \rrbracket^{\mathrm{H}} e) \mid !\llbracket \lambda x.x \rrbracket^{\mathrm{H}} y \mid b \rightarrow a) \\
 & \rightarrow_{\pi}(y,b), \rightsquigarrow_{\mathrm{G}} & (\nu xe)\,(\llbracket x \rrbracket^{\mathrm{H}} e \mid \overline{a}\langle x,e\rangle) \\
 & ? & \llbracket \lambda x.x \rrbracket^{\mathrm{H}} a
\end{array}$$

which clearly shows that we generate an asynchronous output in the interpretation for the result of the reduction. To obtain a translation that respects reduction, we therefore had to define it using asynchronous output for the abstraction. This then, fortunately, opened up the possibility to model reduction under abstraction as well.

## 5.1 Preservation of explicit head reduction

In this section, we will show that our translation respects explicit head reduction. By the very nature of that translation this result is not exactly *" If $M \rightarrow_{\mathrm{xH}} N$, then $\llbracket M \rrbracket^{\mathrm{H}} p \rightarrow_{\pi}^{+} \llbracket N \rrbracket^{\mathrm{H}} p$ "* but instead gets formulated via a relation that also permits renaming of output and removal of mute processes, *i.e.* garbage collection. The renaming equivalence is defined and justified via Lemma 5.6, which states that we can safely rename the output of an interpreted $\lambda\mathbf{x}$-term.

First, we need the following:

*Lemma 5.5* $(\nu a)\,(c(v,d).\,(!\llbracket M \rrbracket^{\mathrm{H}} v \mid d \rightarrow a) \mid a \rightarrow e) \;\sim_{\mathrm{C}}\; c(v,d).\,(\nu a)\,(!\llbracket M \rrbracket^{\mathrm{H}} v \mid d \rightarrow a \mid a \rightarrow e)$.

*Proof:* Any context that interacts with either the left or the right-hand side has to do so via $\overline{c}\langle b,g\rangle$; in both cases this yields the process $(\nu a)\,(!\llbracket M \rrbracket^{\mathrm{H}} b \mid g \rightarrow a \mid a \rightarrow e)$. □

We use this Lemma to show that we can safely redirect the unique output of interpreted lambda terms:

*Lemma 5.6* (Renaming lemma) $(\nu a)\,(\llbracket N \rrbracket^{\mathrm{H}} a \mid a \rightarrow e) \;\sim_{\mathrm{C}}\; \llbracket N \rrbracket^{\mathrm{H}} e$.

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

$(N = x):\;\; (\nu a)\,(\llbracket x \rrbracket^{\mathrm{H}} a \mid a \rightarrow e) \;\stackrel{\Delta}{=}\; (\nu a)\,(x(w).\,\overline{a}\langle w\rangle \mid a(w).\,\overline{e}\langle w\rangle) \;\sim_{\mathrm{C}}\; x(w).\,\overline{e}\langle w\rangle \;\stackrel{\Delta}{=}\; \llbracket x \rrbracket^{\mathrm{H}} e$

$$\begin{array}{rcl}
(N = \lambda x.M):\;\; (\nu a)\,(\llbracket \lambda x.M \rrbracket^{\mathrm{H}} a \mid a \rightarrow e) & \stackrel{\Delta}{=} & \\
(\nu a)\,((\nu xb)\,(\llbracket M \rrbracket^{\mathrm{H}} b \mid \overline{a}\langle x,b\rangle) \mid a \rightarrow e) & \rightarrow_{\pi} & (a \notin \llbracket M \rrbracket^{\mathrm{H}} b) \\
(\nu xb)\,(\llbracket M \rrbracket^{\mathrm{H}} b \mid \overline{e}\langle x,b\rangle) & \stackrel{\Delta}{=} & \llbracket \lambda x.M \rrbracket^{\mathrm{H}} e
\end{array}$$

$$\begin{array}{rcl}
(N = PQ):\;\; (\nu a)\,(\llbracket PQ \rrbracket^{\mathrm{H}} a \mid a \rightarrow e) & \stackrel{\Delta}{=} & \\
(\nu a)\,((\nu c)\,(\llbracket P \rrbracket^{\mathrm{H}} c \mid c(v,d).\,(!\llbracket Q \rrbracket^{\mathrm{H}} v \mid d \rightarrow a)) \mid a \rightarrow e) & \equiv & \\
(\nu c)\,(\llbracket P \rrbracket^{\mathrm{H}} c \mid (\nu a)\,(c(v,d).\,(!\llbracket Q \rrbracket^{\mathrm{H}} v \mid d \rightarrow a) \mid a \rightarrow e)) & \sim_{\mathrm{C}} & (5.5) \\
(\nu c)\,(\llbracket P \rrbracket^{\mathrm{H}} c \mid c(v,d).\,(\nu a)\,(a \rightarrow e \mid !\llbracket Q \rrbracket^{\mathrm{H}} v \mid d \rightarrow a)) & \sim_{\mathrm{C}} & \\
(\nu c)\,(\llbracket P \rrbracket^{\mathrm{H}} c \mid c(v,d).\,(!\llbracket Q \rrbracket^{\mathrm{H}} v \mid d \rightarrow e)) & \stackrel{\Delta}{=} & \llbracket PQ \rrbracket^{\mathrm{H}} e
\end{array}$$

$$\begin{array}{rcl}
(N = M\langle x := L\rangle):\;\; (\nu a)\,(\llbracket M\langle x := L\rangle \rrbracket^{\mathrm{H}} a \mid a \rightarrow e) & \stackrel{\Delta}{=} & \\
(\nu a)\,((\nu x)\,(\llbracket M \rrbracket^{\mathrm{H}} a \mid !\llbracket L \rrbracket^{\mathrm{H}} x) \mid a \rightarrow e) & \equiv & \\
(\nu x)\,((\nu a)\,(\llbracket M \rrbracket^{\mathrm{H}} a \mid a \rightarrow e) \mid \llbracket x := L \rrbracket^{\mathrm{H}}) & \sim_{\mathrm{C}} & (IH) \\
(\nu x)\,(\llbracket M \rrbracket^{\mathrm{H}} e \mid !\llbracket L \rrbracket^{\mathrm{H}} x) & \stackrel{\Delta}{=} & \llbracket M\langle x := L\rangle \rrbracket^{\mathrm{H}} e
\end{array}$$

□

Notice that, in the second part, reduction takes place over a private channel, so the processes involved are contextually equivalent by Proposition 2.5.

**Definition 5.7** *i*) We will use $\rightsquigarrow_{\mathrm{R}}$ if we want to emphasise that two processes are equivalent just using renaming, *i.e.* define: $(\nu a)\,(P \mid a{\rightarrow}e) \;\rightsquigarrow_{\mathrm{R}}\; P[e/a]$ if $a$ is the only free output of $P$ and is only used to output.

*ii*) We define $\rightsquigarrow^*_{\pi}$ as $\rightarrow^*_{\pi} \cup \rightsquigarrow^*_{\mathrm{R}} \cup \rightsquigarrow^*_{\mathrm{G}}$.

Notice that $\rightsquigarrow_{\mathrm{R}} \subset \sim_{\mathrm{C}}$; again, we use an arrow notation to indicate that we will not use this equivalence in the opposite direction. Since we will use renaming only on bound names, this step corresponds to an explicit $\alpha$-conversion. More importantly, in particular, $(\nu a)\,(\llbracket N \rrbracket^{\mathrm{H}} a \mid a{\rightarrow}e) \rightsquigarrow_{\mathrm{R}} \llbracket N \rrbracket^{\mathrm{H}} e$.

Using Lemma 5.6, we can show the following:

*Example 5.8* The translation of a $\beta$-redex $(\lambda x.P)Q$ will yield a process containing a synchronisation that receives on the input channel called $x$ in the interpretation of $P$, and the interpretation of $Q$ being output on $x$ (see Theorem 5.9).

$$\begin{array}{rcll}
\llbracket (\lambda x.P)Q \rrbracket^{\mathrm{H}} a & \triangleq & (\nu c)\,((\nu x b)\,(\llbracket P \rrbracket^{\mathrm{H}} b \mid \overline{c}\langle x,b\rangle) \mid c(v,d).\,(!\llbracket Q \rrbracket^{\mathrm{H}} v \mid d{\rightarrow}a)) & \\
 & \rightarrow_{\pi}(c) & (\nu b x)\,(\llbracket P \rrbracket^{\mathrm{H}} b \mid b{\rightarrow}a \mid !\llbracket Q \rrbracket^{\mathrm{H}} x) & \\
 & \rightsquigarrow_{\mathrm{R}} & (\nu x)\,(\llbracket P \rrbracket^{\mathrm{H}} a \mid !\llbracket Q \rrbracket^{\mathrm{H}} x) & \triangleq\; \llbracket P\langle x{:=}Q\rangle \rrbracket^{\mathrm{H}} a
\end{array}$$

This implies that $\beta$-reduction is implemented in $\pi$ by at least one synchronisation in $\pi$.

Notice that this example justifies the last clause of the definition of our interpretation in Definition 5.1.

As in [25, 30, 32], we can now show a reduction preservation result for explicit head reduction for $\lambda\mathbf{x}$, by showing that $\llbracket \cdot \rrbracket^{\mathrm{H}} \cdot$ preserves $\rightarrow_{\mathrm{xH}}$ up to $\rightsquigarrow^*_{\pi}$. Since reduction in interpreted terms takes place over hidden channels exclusively, we have $\rightsquigarrow^*_{\pi} \subseteq \sim_{\mathrm{C}}$ and could have shown the following result using $\sim_{\mathrm{C}}$. However, the current formulation is more expressive since it states exactly how interpreted terms are related; notice that again we do not require the terms to be closed.

We will now show that our interpretation respects explicit head reduction.

**Theorem 5.9** ($\llbracket \cdot \rrbracket^{\mathrm{H}} \cdot$ PRESERVES $\rightarrow_{\mathrm{xH}}$) *If $M \rightarrow_{\mathrm{xH}} N$, then $\llbracket M \rrbracket^{\mathrm{H}} a \rightsquigarrow^*_{\pi} \llbracket N \rrbracket^{\mathrm{H}} a$.*

*Proof:* By induction on the definition of $\rightarrow_{\mathrm{xH}}$, of which we show only the interesting base cases.

$((\lambda x.M)N \rightarrow M\langle x{:=}N\rangle)$ : By Example 5.8.

$$\begin{array}{rcl}
((\lambda y.M)\langle x{:=}N\rangle \rightarrow \lambda y.(M\langle x{:=}N\rangle)) : \; \llbracket (\lambda y.M)\langle x{:=}N\rangle \rrbracket^{\mathrm{H}} a & \triangleq & \\
(\nu x)\,((\nu y b)\,(\llbracket M \rrbracket^{\mathrm{H}} b \mid \overline{a}\langle y,b\rangle) \mid !\llbracket N \rrbracket^{\mathrm{H}} x) & \equiv & \\
(\nu y b)\,((\nu x)\,(\llbracket M \rrbracket^{\mathrm{H}} b \mid !\llbracket N \rrbracket^{\mathrm{H}} x) \mid \overline{a}\langle y,b\rangle) & \triangleq & \llbracket \lambda y.M\langle x{:=}N\rangle \rrbracket^{\mathrm{H}} a
\end{array}$$

$$\begin{array}{rcl}
(x\overrightarrow{M}\overrightarrow{\langle y{:=}L\rangle}\langle x{:=}N\rangle \rightarrow N\overrightarrow{M}\overrightarrow{\langle y{:=}L\rangle}\langle x{:=}N\rangle) : \; \llbracket x\overrightarrow{M}\overrightarrow{\langle y{:=}L\rangle}\langle x{:=}N\rangle \rrbracket^{\mathrm{H}} a & & \triangleq \\
(\nu x\vec{y})\,(\llbracket x\overrightarrow{M} \rrbracket^{\mathrm{H}} a \mid !\overrightarrow{\llbracket L \rrbracket^{\mathrm{H}} y} \mid !\llbracket N \rrbracket^{\mathrm{H}} x) & & \triangleq \\
(\nu x\vec{y})\,((\nu c_n)\,(\llbracket x M_1 \cdots M_{n-1} \rrbracket^{\mathrm{H}} c_n \mid c_n(v,d).\,(!\llbracket M_n \rrbracket^{\mathrm{H}} v \mid d{\rightarrow}a)) \mid !\overrightarrow{\llbracket L \rrbracket^{\mathrm{H}} y} \mid !\llbracket N \rrbracket^{\mathrm{H}} x) & & \triangleq, \equiv \\
(\nu x\vec{y})\,((\nu c_n \cdots c_1)\,(x(w).\overline{c_1}\langle w\rangle \mid c_1(b,d).\,(!\llbracket M_1 \rrbracket^{\mathrm{H}} b \mid d{\rightarrow}c_2) \mid \cdots \mid & & \\
\qquad c_n(b,d).\,(!\llbracket M_n \rrbracket^{\mathrm{H}} b \mid d{\rightarrow}a)) \mid !\overrightarrow{\llbracket L \rrbracket^{\mathrm{H}} y} \mid \llbracket N \rrbracket^{\mathrm{H}} x \mid !\llbracket N \rrbracket^{\mathrm{H}} x) & & \rightsquigarrow_{\mathrm{R}}(x) \\
(\nu x\vec{y})\,((\nu c_n \cdots c_1)\,(\llbracket N \rrbracket^{\mathrm{H}} c_1 \mid c_1(b,d).\,(!\llbracket M_1 \rrbracket^{\mathrm{H}} b \mid d{\rightarrow}c_2) \mid \cdots \mid & & \\
\qquad c_n(b,d).\,(!\llbracket M_n \rrbracket^{\mathrm{H}} b \mid d{\rightarrow}a)) \mid !\overrightarrow{\llbracket L \rrbracket^{\mathrm{H}} y} \mid !\llbracket N \rrbracket^{\mathrm{H}} x) & & \triangleq \\
\llbracket N M_1 \cdots M_m \overrightarrow{\langle y{:=}L\rangle}\langle x{:=}N\rangle \rrbracket^{\mathrm{H}} a & &
\end{array}$$

$$
\begin{array}{ll}
\llbracket (\lambda x.xx)(\lambda y.y) \rrbracket^{\mathtt{H}}_{\mathbb{I}} a & \stackrel{\Delta}{=} \\
(\nu c)\,((\nu xb)\,(\llbracket xx \rrbracket^{\mathtt{H}}_{\mathbb{I}} b \mid \overline{c}\langle x,b\rangle) \mid c(v,d).\,(!\llbracket \lambda y.y \rrbracket^{\mathtt{H}}_{\mathbb{I}} v \mid d \twoheadrightarrow a)) & \to_{\pi} (c), \rightsquigarrow_{\mathrm{R}} \\
(\nu x)\,(\llbracket xx \rrbracket^{\mathtt{H}}_{\mathbb{I}} a \mid !\llbracket \lambda y.y \rrbracket^{\mathtt{H}}_{\mathbb{I}} x) & \stackrel{\Delta}{=} \\
(\nu x)\,((\nu c)\,(x(w).\overline{c}\langle w\rangle \mid c(v,d).\,(!\llbracket x \rrbracket^{\mathtt{H}}_{\mathbb{I}} v \mid d \twoheadrightarrow a)) \mid !(\nu yb)\,(\llbracket y \rrbracket^{\mathtt{H}}_{\mathbb{I}} b \mid \overline{x}\langle y,b\rangle)) & \to_{\pi} (x) \\
(\nu x)\,((\nu c)\,((\nu zb)\,(\llbracket z \rrbracket^{\mathtt{H}}_{\mathbb{I}} b \mid \overline{c}\langle z,b\rangle) \mid c(v,d).\,(!\llbracket x \rrbracket^{\mathtt{H}}_{\mathbb{I}} v \mid d \twoheadrightarrow a)) \mid !\llbracket \lambda y.y \rrbracket^{\mathtt{H}}_{\mathbb{I}} x) & \to_{\pi} (c), \rightsquigarrow_{\mathrm{R}} \\
(\nu x)\,((\nu z)\,(\llbracket z \rrbracket^{\mathtt{H}}_{\mathbb{I}} a \mid !\llbracket x \rrbracket^{\mathtt{H}}_{\mathbb{I}} z) \mid !\llbracket \lambda y.y \rrbracket^{\mathtt{H}}_{\mathbb{I}} x) & \equiv, \stackrel{\Delta}{=} \\
(\nu x)\,((\nu z)\,(z(w).\overline{a}\langle w\rangle \mid x(w).\overline{z}\langle w\rangle \mid !\llbracket x \rrbracket^{\mathtt{H}}_{\mathbb{I}} z) \mid !\llbracket \lambda y.y \rrbracket^{\mathtt{H}}_{\mathbb{I}} x) & \rightsquigarrow_{\mathrm{R}} \\
(\nu x)\,((\nu z)\,(x(w).\overline{a}\langle w\rangle \mid !\llbracket x \rrbracket^{\mathtt{H}}_{\mathbb{I}} z) \mid !\llbracket \lambda y.y \rrbracket^{\mathtt{H}}_{\mathbb{I}} x) & \rightsquigarrow_{\mathrm{G}} \\
(\nu x)\,(x(w).\overline{a}\langle w\rangle \mid !(\nu yb)\,(\llbracket y \rrbracket^{\mathtt{H}}_{\mathbb{I}} b \mid \overline{x}\langle y,b\rangle)) & \to_{\pi} (x) \\
(\nu x)\,((\nu zb)\,(\llbracket z \rrbracket^{\mathtt{H}}_{\mathbb{I}} b \mid \overline{a}\langle z,b\rangle) \mid !\llbracket \lambda y.y \rrbracket^{\mathtt{H}}_{\mathbb{I}} x) & \stackrel{\Delta}{=}, \rightsquigarrow_{\mathrm{G}} \quad \llbracket \lambda z.z \rrbracket^{\mathtt{H}}_{\mathbb{I}} a
\end{array}
$$

Figure 3: Running $\llbracket (\lambda x.xx)(\lambda y.y) \rrbracket^{\mathtt{H}}_{\mathbb{I}} a$ with renaming.

$$
\begin{array}{ll}
(M\langle x := N\rangle \to M,\ x \notin fv(M)):\ \llbracket M\langle x := N\rangle \rrbracket^{\mathtt{H}}_{\mathbb{I}} a & \stackrel{\Delta}{=} \\
\quad (\nu x)\,(\llbracket M \rrbracket^{\mathtt{H}}_{\mathbb{I}} a \mid !\llbracket N \rrbracket^{\mathtt{H}}_{\mathbb{I}} x) & \equiv \\
\quad \llbracket M \rrbracket^{\mathtt{H}}_{\mathbb{I}} a \mid (\nu x)\,(!\llbracket N \rrbracket^{\mathtt{H}}_{\mathbb{I}} x) & \rightsquigarrow_{\mathrm{G}} \ \llbracket M \rrbracket^{\mathtt{H}}_{\mathbb{I}} a
\end{array}
\qquad \square
$$

So, perhaps contrary to expectation, because abstraction is not translated using *input*, we can without problem model reduction, modulo renaming, under a $\lambda$-abstraction.

Notice that, in this proof, reduction is only used in the first case, and that renaming (which might involve reduction) is used in the first and third case (notice that there we use $(\nu x)\,(\llbracket N \rrbracket^{\mathtt{H}}_{\mathbb{I}} x \mid x(w).\overline{c_1}\langle w\rangle) \rightsquigarrow_{\mathrm{R}} \llbracket N \rrbracket^{\mathtt{H}}_{\mathbb{I}} c_1$); the other cases are dealt with by congruence and garbage collection. Also, notice that by Proposition 5.2 the process $(\nu x)\,(!\llbracket N \rrbracket^{\mathtt{H}}_{\mathbb{I}} x)$ is *mute* (contextually equivalent to $\mathtt{0}$, and therefore garbage), a property that is used in the fourth case of the proof above. Moreover, as for $\to_{\mathrm{xL}}$, garbage collection is only used to model $(\mathtt{gc})$.

*Example 5.10* In Example 3.7, we showed $(\lambda x.xx)(\lambda y.y) \to^{*}_{\mathrm{xH}} \lambda z.z$. Then, by repeatedly applying Theorem 5.9,

$$
\begin{array}{rll}
\llbracket (\lambda x.xx)(\lambda y.y) \rrbracket^{\mathtt{H}}_{\mathbb{I}} a & \rightsquigarrow^{*}_{\pi} & \llbracket xx\langle x := \lambda y.y\rangle \rrbracket^{\mathtt{H}}_{\mathbb{I}} a \\
& \rightsquigarrow^{*}_{\pi} & \llbracket (\lambda z.z)x\langle x := \lambda y.y\rangle \rrbracket^{\mathtt{H}}_{\mathbb{I}} a \\
& \rightsquigarrow^{*}_{\pi} & \llbracket z\langle z := x\rangle\langle x := \lambda y.y\rangle \rrbracket^{\mathtt{H}}_{\mathbb{I}} a \\
& \rightsquigarrow^{*}_{\pi} & \llbracket x\langle z := x\rangle\langle x := \lambda y.y\rangle \rrbracket^{\mathtt{H}}_{\mathbb{I}} a \\
& \rightsquigarrow^{*}_{\pi} & \llbracket x\langle x := \lambda y.y\rangle \rrbracket^{\mathtt{H}}_{\mathbb{I}} a \\
& \rightsquigarrow^{*}_{\pi} & \llbracket (\lambda z.z)\langle x := \lambda y.y\rangle \rrbracket^{\mathtt{H}}_{\mathbb{I}} a \qquad \rightsquigarrow^{*}_{\pi} \ \llbracket \lambda z.z \rrbracket^{\mathtt{H}}_{\mathbb{I}} a
\end{array}
$$

Figure 3 give the details of this reduction; notice that it uses renaming which was not used in Figure 2. Mark that the first and third $(\mathtt{App})$-step are executed through $\to_{\pi}$, and the second through renaming.

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

$$
(\nu c)\,(\llbracket \lambda y.y \rrbracket^{\mathtt{H}}_{\mathbb{I}} c \mid c(v,d).\,(!\llbracket \lambda y.y \rrbracket^{\mathtt{H}}_{\mathbb{I}} v \mid d \twoheadrightarrow a)) \ \stackrel{\Delta}{=}\ \llbracket (\lambda y.y)(\lambda y.y) \rrbracket^{\mathtt{H}}_{\mathbb{I}} a.
$$

We can now show operational soundness.

**Theorem 5.11** (Operational Soundness for $\to_{\mathrm{xH}}$) *i ) If $M \to^{*}_{\mathrm{xH}} N$, then $\llbracket M \rrbracket^{\mathtt{H}}_{\mathbb{I}} a \rightsquigarrow^{*}_{\pi} \llbracket N \rrbracket^{\mathtt{H}}_{\mathbb{I}} a$.*
*ii ) If $M \to^{\not{c}*}_{\mathrm{xH}} N$, then $\llbracket M \rrbracket^{\mathtt{H}}_{\mathbb{I}} a \to^{*}_{\pi}, \rightsquigarrow^{*}_{\mathrm{R}} \llbracket N \rrbracket^{\mathtt{H}}_{\mathbb{I}} a$.*
*iii ) If $M\uparrow_{\mathrm{xH}}$, then $\llbracket M \rrbracket^{\mathtt{H}}_{\mathbb{I}} a\uparrow$.*

*Proof:* The first is shown by induction on the length of reduction sequences using Proposition 3.5 and Theorem 5.9; the third follows from the observation that each infinite $\to_{\mathrm{xH}}$

reduction sequence has infinitely many $(\beta)$ steps, and each of these corresponds to at least one $\pi$-synchronisation step, as shown in Example 5.8. $\square$

Notice that the second case states the result for $\rightarrow_{\mathbf{xH}}^{\not{\mathsf{gc}}}$ and shows that garbage collection in the $\pi$-calculus is only needed to model the rule $(\mathsf{gc})$ of $\rightarrow_{\mathbf{xH}}$.

Theorem 5.11 directly leads to:

*Corollary 5.12 If* $M \rightarrow_{\mathbf{xH}}^{*} (\lambda x.N)\,\overline{\langle y{:=}L\rangle}$, *then* $\llbracket M \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, a \rightsquigarrow_{\pi}^{*} (\nu\vec{y})\,(\llbracket \lambda x.N \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, a \mid !\,\overline{\llbracket L \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, y})$.

a reformulation of Milner's result.

Since lazy reduction is included in head reduction, we also have:

*Corollary 5.13 If* $M \rightarrow_{\mathbf{xL}}^{*} N$, *then* $\llbracket M \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, a \rightsquigarrow_{\pi}^{*} \llbracket N \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, a$.

Since $\rightsquigarrow_{\pi}^{*} \subseteq \sim_{\mathrm{C}}$, and $\sim_{\mathrm{C}}$ is symmetric, we immediately get the following:

*Corollary 5.14 i ) If* $M \rightarrow_{\mathbf{xH}}^{*} N$, *then* $\llbracket M \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, a \sim_{\mathrm{C}} \llbracket N \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, a$.
*ii ) If* $M =_{\mathbf{xH}} N$, *then* $\llbracket M \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, a \sim_{\mathrm{C}} \llbracket N \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, a$.

which states that, in fact, our encoding gives a semantics for explicit head reduction.

As to an operational completeness result "*If* $\llbracket M \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, a \rightarrow_{\pi} Q$, *then there exists* $N$ *such that* $M \rightarrow_{\mathbf{xH}}^{*} N$, *and* $Q \rightarrow_{\pi}^{*} R$ *and* $R \sim_{\mathrm{C}} \llbracket N \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, a$", this does not hold in general, as illustrated by the next example.

*Example 5.15* Observe that
$$\begin{array}{ll} (\lambda x.(\lambda y.zN)(xM))(\lambda u.L) & \rightarrow_{\mathbf{xH}} \\ (\lambda y.zN)(xM)\,\langle x{:=}\lambda u.L\rangle & \rightarrow_{\mathbf{xH}} \\ zN\,\langle y{:=}xM\rangle\,\langle x{:=}\lambda u.L\rangle & \end{array}$$
and that, assuming $z \neq y, x$, this last term is in $\rightarrow_{\mathbf{xH}}$-normal form. As by Theorem 5.11, we have
$$\llbracket (\lambda x.(\lambda y.zN)(xM))(\lambda u.L) \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, a \;\rightsquigarrow_{\pi}^{*}\; \llbracket zN\,\langle y{:=}xM\rangle\,\langle x{:=}\lambda u.L\rangle \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, a$$

However, we can now continue reduction in a way not intended by our explicit head reduction:

$$\begin{array}{ll} \llbracket zN\,\langle y{:=}xM\rangle\,\langle x{:=}\lambda u.L\rangle \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, a & \triangleq \\ (\nu x)\,((\nu y)\,(\llbracket zN \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, a \mid !\llbracket xM \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, y) \mid !\llbracket \lambda u.L \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, x) & \equiv \\ (\nu xy)\,(\llbracket zN \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, a \mid \llbracket xM \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, y \mid \llbracket \lambda u.L \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, x \mid !\llbracket xM \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, y \mid !\llbracket \lambda u.L \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, x) & \triangleq \\ (\nu xy)\,(\llbracket zN \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, a \mid (\nu c)\,(x(w).\,\overline{c}\langle w\rangle \mid c(v,d).\,(!\llbracket M \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, v \mid d{\rightarrow}y)) \mid (\nu ub)\,(\llbracket L \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, b \mid \overline{x}\langle u,b\rangle) \mid & \\ \qquad\qquad !\llbracket xM \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, y \mid !\llbracket \lambda u.L \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, x) & \rightarrow_{\pi} (x) \\ (\nu xy)\,(\llbracket zN \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, a \mid (\nu c)\,((\nu ub)\,(\llbracket L \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, b \mid \overline{c}\langle u,b\rangle) \mid c(v,d).\,(!\llbracket M \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, v \mid d{\rightarrow}y)) \mid !\llbracket xM \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, y \mid !\llbracket \lambda u.L \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, x) & \triangleq \\ (\nu xy)\,(\llbracket zN \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, a \mid \llbracket (\lambda u.L)M \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, y \mid !\llbracket xM \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, y \mid !\llbracket \lambda u.L \rrbracket^{\mathbf{H}}_{\mathbf{J}}\, x) & \end{array}$$

which does not correspond to a $\rightarrow_{\mathbf{xH}}$-reduction.

We can only show the desired completeness results when blocking this kind of synchronisation step and will address this issue in the next section.

## 5.2 On Termination

In the context of encodings, since the notions of reduction vary greatly between various calculi, it comes as no surprise that normally not all the desired properties are preserved. For example, the preservation of termination "*if* $M$ *terminates, then so does* $\llbracket M \rrbracket$" cannot always be shown. Take, for example, the $\lambda$-calculus and Combinatory Logic (CL) [16]; although these are naturally linked, the fact that reduction is strong in the first and weak in the second

makes preservation of termination complicated. Note that although the encoding of the $\lambda$-calculus into CL (for details, see [10]) is well behaved in that reduction is preserved, the reverse encoding is not complete nor preserves termination:

*Example 5.16* ([10]) *i*) $t = \mathbf{S}(\mathbf{K}(\mathbf{SII}))(\mathbf{K}(\mathbf{SII}))$ is a normal form, but $[\![t]\!]_\lambda \to_\beta^+ \lambda c.(\lambda x.xx)(\lambda x.xx)$, which does not have a $\beta$-normal form, and not even a head-normal from, so no observable behaviour.

*ii*) $t = \mathbf{SK}((\mathbf{SII})(\mathbf{SII}))$ has no normal form, while $[\![t]\!]_\lambda \to_\beta^+ \lambda x.x$.

(See [6] for a discussion on approximation semantics and full abstraction for Combinatory Systems).

For another example, neither the encoding of recursive programs into the $\lambda$-calculus is terminating; 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 parts. This could be solved by enforcing a reduction strategy on the target language, typically via lazy reduction, but at the price that, as argued above, Church numerals are no longer considered a good encoding for numbers. It should be clear that this does not imply that the $\lambda$-calculus is not a proper model of computation, so termination is not a decisive criterion on the suitability of an encoding.

We encounter the termination problem with translations of the $\lambda$-calculus into the $\pi$-calculus: as argued above, to model implicit substitution replication has to be used, which potentially introduces non-termination. For Milner's encoding, termination causes no difficulties, In fact, it solves the termination problem because the interpretation of the substitution of $N$ is placed under guard (as in $!\,x(w).\,[\![N]\!]^{\mathrm{M}}_{\mathrm{J}}\,w$) which blocks the running of arbitrarily many copies; the synchronisation over $x$ is the deblocking action for a peeled-off copy. Essentially, termination is achieved because each reduction step taken to reach the lazy normal form corresponds to a finite number of synchronisation steps in the $\pi$-calculus, and in the created process

$$(\overline{\nu x})\,([\![\lambda y.R]\!]^{\mathrm{M}}_{\mathrm{J}}\,u \mid [\![\overrightarrow{x := N}]\!]^{\mathrm{M}}_{\mathrm{J}}) \;=\; (\overline{\nu x})\,(u(y).\,u(v).\,[\![R]\!]^{\mathrm{M}}_{\mathrm{J}}\,v \mid \overrightarrow{!\,x(w).\,[\![N]\!]^{\mathrm{M}}_{\mathrm{J}}\,w})$$

no synchronisation is possible inside $[\![R]\!]^{\mathrm{M}}_{\mathrm{J}}\,v$ or $[\![N]\!]^{\mathrm{M}}_{\mathrm{J}}\,w$, since these are both placed under *input*; the termination result comes, therefore, at the price of restricting the interpreted reduction on $\lambda$-terms to the large-step reduction to normal form of the lazy $\lambda$-calculus.

Notice that Theorem 5.11 shows that under $\llbracket\cdot\rrbracket^{\mathrm{H}}_{\mathrm{J}}\cdot$ the internal reduction in the image of $\lambda$-terms is unobservable, and that the translation of a term is (renaming, garbage collected) equivalent to the translation of its normal form. Since internal reductions are not observable, this does of course not preclude that the translation perhaps creates an infinite amount of reductions that do not correspond to reduction steps in $\lambda\mathbf{x}$; and, in fact, since replication is unguarded, inside the translation of the explicit substitution $\llbracket M\langle x := N\rangle\rrbracket^{\mathrm{H}}_{\mathrm{J}}\,a \triangleq (\nu x)\,(\llbracket M\rrbracket^{\mathrm{H}}_{\mathrm{J}}\,a \mid !\llbracket N\rrbracket^{\mathrm{H}}_{\mathrm{J}}\,x)$, infinitely many copies of $\llbracket N\rrbracket^{\mathrm{H}}_{\mathrm{J}}\,x$ are free to run, each exposing no observable behaviour since $x$ is bound, but this introduces a form of non-termination and no longer simulates the interpreted term; since the $\pi$-calculus has no notion of *erasure*, this will always leave a non-terminating part.

So is this problematic? By the proofs shown in the previous section, clearly the replication issue does not affect our translation results, but it could be problematic when we aim for termination, *i.e.* if we want to show that a terminating explicit head reduction is interpreted via a terminating reduction.

We can achieve termination easily by changing the interpretation through the placement, as for Milner's, of a guard on the replication.

$$
\begin{array}{ll}
\llbracket(\lambda x.xx)(\lambda q.q)\rrbracket^{\mathrm{T}}_{\mathrm{J}}a & \stackrel{\Delta}{=},\equiv \\
(\nu c)\,((\nu xb)\,(\llbracket xx\rrbracket^{\mathrm{T}}_{\mathrm{J}}b \mid \overline{c}\langle x,b\rangle) \mid c(v,d).\,(\llbracket v := \lambda q.q\rrbracket^{\mathrm{T}}_{\mathrm{J}} \mid d\rightarrow a)) & \rightarrow_{\pi}(c) \\
(\nu xb)\,(\llbracket xx\rrbracket^{\mathrm{T}}_{\mathrm{J}}b \mid \llbracket x := \lambda q.q\rrbracket^{\mathrm{T}}_{\mathrm{J}} \mid b\rightarrow a) & \stackrel{\Delta}{=} \\
(\nu xb)\,((\nu c)\,(x(z).\,z\rightarrow c \mid c(v,d).\,(\llbracket v := x\rrbracket^{\mathrm{T}}_{\mathrm{J}} \mid d\rightarrow b)) \mid !\,(\nu w)\,(\overline{x}\langle w\rangle.\,\llbracket\lambda q.q\rrbracket^{\mathrm{T}}_{\mathrm{J}}w) \mid b\rightarrow a) & \rightarrow_{\pi}(x)\,(y/q) \\
(\nu xb)\,((\nu w)\,((\nu c)\,(w\rightarrow c \mid c(v,d).\,(\llbracket v := x\rrbracket^{\mathrm{T}}_{\mathrm{J}} \mid d\rightarrow b)) \mid & \\
\qquad (\nu yb')\,(\llbracket y\rrbracket^{\mathrm{T}}_{\mathrm{J}}b' \mid \overline{w}\langle y,b'\rangle))) \mid \llbracket x := \lambda q.q\rrbracket^{\mathrm{T}}_{\mathrm{J}} \mid b\rightarrow a) & \rightarrow_{\pi}(w) \\
(\nu xb)\,((\nu c)\,((\nu yb')\,(\llbracket y\rrbracket^{\mathrm{T}}_{\mathrm{J}}b' \mid \overline{c}\langle y,b'\rangle) \mid c(v,d).\,(\llbracket v := x\rrbracket^{\mathrm{T}}_{\mathrm{J}} \mid d\rightarrow b)) \mid \llbracket x := \lambda q.q\rrbracket^{\mathrm{T}}_{\mathrm{J}} \mid b\rightarrow a) & \rightarrow_{\pi}(c) \\
(\nu xb)\,((\nu yb')\,(\llbracket y\rrbracket^{\mathrm{T}}_{\mathrm{J}}b' \mid \llbracket y := x\rrbracket^{\mathrm{T}}_{\mathrm{J}} \mid b'\rightarrow b) \mid \llbracket x := \lambda q.q\rrbracket^{\mathrm{T}}_{\mathrm{J}} \mid b\rightarrow a) & \stackrel{\Delta}{=} \\
(\nu xb)\,((\nu yb')\,(y(z).\,z\rightarrow b' \mid !\,(\nu w)\,(\overline{y}\langle w\rangle.\,\llbracket x\rrbracket^{\mathrm{T}}_{\mathrm{J}}w) \mid b'\rightarrow b) \mid \llbracket x := \lambda q.q\rrbracket^{\mathrm{T}}_{\mathrm{J}} \mid b\rightarrow a) & \rightarrow_{\pi}(y) \\
(\nu xb)\,((\nu yb')\,((\nu w')\,(w'\rightarrow b' \mid x(z).\,z\rightarrow w') \mid \llbracket y := x\rrbracket^{\mathrm{T}}_{\mathrm{J}} \mid b'\rightarrow b) \mid & \\
\qquad !\,(\nu w)\,(\overline{x}\langle w\rangle.\,(\nu rb'')\,(\llbracket r\rrbracket^{\mathrm{T}}_{\mathrm{J}}b'' \mid \overline{w}\langle r,b''\rangle))) \mid b\rightarrow a) & \rightarrow_{\pi}(x) \\
(\nu xb)\,((\nu yb')\,((\nu w)\,((\nu w')\,(w'\rightarrow b' \mid w\rightarrow w') \mid \llbracket y := x\rrbracket^{\mathrm{T}}_{\mathrm{J}} \mid b'\rightarrow b)) \mid & \\
\qquad (\nu rb'')\,(\llbracket r\rrbracket^{\mathrm{T}}_{\mathrm{J}}b'' \mid \overline{w}\langle r,b''\rangle) \mid \llbracket x := \lambda q.q\rrbracket^{\mathrm{T}}_{\mathrm{J}} \mid b\rightarrow a) & \rightarrow_{\pi}(w,w',b',b) \\
(\nu rb'')\,(\llbracket r\rrbracket^{\mathrm{T}}_{\mathrm{J}}b'' \mid \overline{a}\langle r,b''\rangle) \mid (\nu xy)\,(\llbracket y := x\rrbracket^{\mathrm{T}}_{\mathrm{J}} \mid \llbracket x := \lambda q.q\rrbracket^{\mathrm{T}}_{\mathrm{J}}) & \sim_{\mathrm{C}} \\
(\nu rb)\,(\llbracket r\rrbracket^{\mathrm{T}}_{\mathrm{J}}b \mid \overline{a}\langle r,b\rangle) & \stackrel{\Delta}{=} \quad \llbracket\lambda r.r\rrbracket^{\mathrm{T}}_{\mathrm{J}}a
\end{array}
$$

Figure 4: Running $\llbracket(\lambda x.xx)(\lambda y.y)\rrbracket^{\mathrm{T}}_{\mathrm{J}}a$.

**Definition 5.17** (Terminating interpretation of the $\lambda$-calculus in $\pi$) The mapping $\llbracket\cdot\rrbracket^{\mathrm{T}}_{\mathrm{J}}\cdot$ is defined by:

$$
\begin{array}{rcll}
\llbracket x\rrbracket^{\mathrm{T}}_{\mathrm{J}}a & \stackrel{\Delta}{=} & x(z).\,z\rightarrow a & x \neq a \\
\llbracket\lambda x.M\rrbracket^{\mathrm{T}}_{\mathrm{J}}a & \stackrel{\Delta}{=} & (\nu xb)\,(\llbracket M\rrbracket^{\mathrm{T}}_{\mathrm{J}}b \mid \overline{a}\langle x,b\rangle) & b \textit{ fresh} \\
\llbracket MN\rrbracket^{\mathrm{T}}_{\mathrm{J}}a & \stackrel{\Delta}{=} & (\nu c)\,(\llbracket M\rrbracket^{\mathrm{T}}_{\mathrm{J}}c \mid c(v,d).\,(\llbracket v := N\rrbracket^{\mathrm{T}}_{\mathrm{J}} \mid d\rightarrow a)) & b,c,d \textit{ fresh} \\
\llbracket x := N\rrbracket^{\mathrm{T}}_{\mathrm{J}} & \stackrel{\Delta}{=} & !\,(\nu w)\,(\overline{x}\langle w\rangle.\,\llbracket N\rrbracket^{\mathrm{T}}_{\mathrm{J}}w) &
\end{array}
$$

Notice that this translation maps into the *synchronous* $\pi$-calculus, but that, as before, abstraction is modelled using an *asynchronous* send. Now running an interpretation needs more steps, as can be seen from Figure 4 and termination is easy to show.

This alternative encoding $\llbracket\cdot\rrbracket^{\mathrm{T}}_{\mathrm{J}}\cdot$ satisfies all properties show above for $\llbracket\cdot\rrbracket^{\mathrm{H}}_{\mathrm{J}}\cdot$, as could be shown using the same techniques as we have used above. The reason we did not present this as our main encoding lies in the fact that $\llbracket\cdot\rrbracket^{\mathrm{H}}_{\mathrm{J}}\cdot$ is the 'natural' encoding that directly follows from the logical foundation, and the use of guards is but a 'fix' to achieve termination, and not needed for any of the simulation results: for example, we would still need renaming.

It is straightforward to show that termination is now preserved; we skip the proof.

**Theorem 5.18** *i ) If $M \rightarrow^{*}_{\mathrm{XH}} N$, with $N$ in $\rightarrow_{\mathrm{XH}}$-normal from, then $\llbracket M\rrbracket^{\mathrm{T}}_{\mathrm{J}}a\downarrow_{\pi}$.*
*ii ) If $M \rightarrow^{*}_{\beta} N$, with $N$ in head-normal from, then $\llbracket M\rrbracket^{\mathrm{T}}_{\mathrm{J}}a\downarrow_{\pi}$.*
*iii ) If $M\downarrow_{\beta}$, then $\llbracket M\rrbracket^{\mathrm{T}}_{\mathrm{J}}a\downarrow_{\pi}$.*

Notice that we can prove a completeness result for this changed encoding:

**Theorem 5.19** (Operational Completeness for $\llbracket\cdot\rrbracket^{\mathrm{T}}_{\mathrm{J}}\cdot$ with respect to $\rightarrow_{\mathrm{XH}}$) *If $\llbracket M\rrbracket^{\mathrm{T}}_{\mathrm{J}}a \rightarrow_{\pi} Q$, then there exists $N$ such that $M \rightarrow^{*}_{\mathrm{XH}} N$, and $Q \rightarrow^{*}_{\pi} R$ and $R \rightsquigarrow^{*}_{\mathrm{R}}, \sim\!\!\rightarrow^{*}_{\mathrm{G}} \llbracket N\rrbracket^{\mathrm{T}}_{\mathrm{J}}a$.*

*Proof:* Notice that synchronisation over $x$ in

$$
\llbracket y := xM\rrbracket^{\mathrm{T}}_{\mathrm{J}} \mid \llbracket x := \lambda u.L\rrbracket^{\mathrm{T}}_{\mathrm{J}} \;\stackrel{\Delta}{=}\; !\,(\nu w)\,(\overline{y}\langle w\rangle.\,\llbracket xM\rrbracket^{\mathrm{T}}_{\mathrm{J}}w) \mid !\,(\nu w)\,(\overline{x}\langle w\rangle.\,\llbracket\lambda u.L\rrbracket^{\mathrm{T}}_{\mathrm{J}}w)
$$

(in Example 5.15 this is $!\llbracket xM\rrbracket^{\mathrm{H}}_{\mathrm{J}}y \mid !\llbracket\lambda u.L\rrbracket^{\mathrm{H}}_{\mathrm{J}}x$) is impossible, since input over $x$ occurs under guard. So reduction in the image only takes place if the 'receiving' channel does not

occur inside a replicated process. Since the only synchronisations possible are those that correspond to a $\rightarrow_{\mathbf{xH}}$-redex, and in that case, also $M$ contracts (to $N$, say); notice that then perhaps renaming and/or garbage collection might be necessary to reach $\llbracket N \rrbracket^{\mathrm{H}} a$. $\square$

### 5.3 Call by Need reduction

An alternative to placing a guard is to limit the reduction engine in the $\pi$-calculus to call by need reduction, much the same way that reduction in the $\lambda$-calculus is limited to lazy reduction to achieve termination of the encoding of computable functions. Another reason is that now the full encoding we present in Section 7 is a variant of our main encoding $\llbracket \cdot \rrbracket^{\mathrm{H}} \cdot$; had we used guards there would be no real relation between the two.

The restriction we introduce is to allow unfolding of $!P$ into $P \mid !P$ only if there is a need to; we see replicated processes as 'sources', into we tap at need, but that are not allowed to start unfolding by themselves. We therefore need to change the nature of the reduction relation through a change in the congruence relation.

**Definition 5.20** (Call by Need reduction) *i*) We call an input name *reachable* in $P$ if it occurs free at and least once not under replication, and is not used for output.

*ii*) We write $P^a$ to express that $a$ is the only free output of $P$, and only used for output.

*iii*) We define the reduction relation $\rightarrow_{\mathrm{N}\pi}$ as in Definition 2.3, but replace '$!P \equiv P \mid !P \equiv !P \mid !P$' in the definition of congruence by

$$Q \mid !P^b \;\equiv_{\mathrm{N}}\; (\nu a)\,(Q[a/b] \mid P[a/b]) \mid !P^b \quad (b \textit{ is reachable in } Q)$$

Whether a name is reachable is decidable. Notice that, in the congruence rule, we need to replace $b$ by a new name $a$ to stop $!P^b$ from unfolding again towards $Q$. In particular, $(\nu x)\,(x(w).\overline{a}\langle w\rangle \mid !\llbracket N \rrbracket^{\mathrm{H}} x) \equiv_{\mathrm{N}} (\nu x)\,((\nu y)\,(y(w).\overline{a}\langle w\rangle \mid \llbracket N \rrbracket^{\mathrm{H}} y) \mid !\llbracket N \rrbracket^{\mathrm{H}} x)$.

We now show that the interpretation of an explicit head-normal form is in call by need normal form.

*Lemma 5.21* (Normal form preservation with respect to $\rightarrow_{\mathrm{N}\pi}$) *Let $\lambda\vec{y}.x\overrightarrow{M}\,\overrightarrow{\langle z{:=}N\rangle}$ be an explicit head-normal form, then the process $\llbracket \lambda\vec{y}.x\overrightarrow{M}\,\overrightarrow{\langle z{:=}N\rangle} \rrbracket^{\mathrm{H}} a$ is in call by need normal form.*

*Proof:* $\llbracket \lambda\vec{y}.x\overrightarrow{M}\,\overrightarrow{\langle z{:=}N\rangle} \rrbracket^{\mathrm{H}} a \;\triangleq$

$$\begin{aligned}(\nu z_k\cdots z_1)\,((\nu y_1 a_1\cdots y_n a_n)\,((\nu c_1\cdots c_m)\,(x(w).\overline{c_m}\langle w\rangle \mid c_m(v,d).\,(!\llbracket M_m \rrbracket^{\mathrm{H}} v \mid d{\rightarrow}c_{m-1}) \mid \cdots \mid \\ c_1(v,d).\,(!\llbracket M_1 \rrbracket^{\mathrm{H}} v \mid d{\rightarrow}a_n)) \mid \overline{a_{n-1}}\langle y_n,a_n\rangle \mid \cdots \mid \overline{a}\langle y_1,a_1\rangle) \mid !\llbracket N_1 \rrbracket^{\mathrm{H}} z_1 \mid \cdots \mid !\llbracket N_k \rrbracket^{\mathrm{H}} z_k)\end{aligned}$$

Now we can make the following observations:

*i*) no synchronisations can take place over any $a_i$: the output $a$ is free, and no process inputs on it, and all $a_i$ can only be used for synchronisation after synchronisation over $a_{i-1}$ (where $a_0 = a$) has taken place, so all are blocked;

*ii*) the replication on $!\llbracket M_i \rrbracket^{\mathrm{H}} v$ cannot unfold since the $v$ is an input variable, so cannot appear free elsewhere.

*iii*) the replication on $!\llbracket N_i \rrbracket^{\mathrm{H}} z_i$ cannot unfold since each $z_i$ occurs (if at all) in a $!\llbracket M_j \rrbracket^{\mathrm{H}} b$, or inside a $!\llbracket N_j \rrbracket^{\mathrm{H}} z_j$, so is never reachable.

So this process is in call by need normal form. $\square$

It is easy to show that in the interpretation of a $\lambda\mathbf{x}$-term, all but the head-variable will occur under replication, so under call by need reduction, processes trying to synchronise on channels that have $\lambda$-variable names are all blocked, but for that communicating with the head-variable. This implies that we can express the result of Theorem 5.9 in terms of call by need reduction.

**Theorem 5.22** ($\llbracket\cdot\rrbracket^{\mathrm{H}}\cdot$ PRESERVES $\to_{\mathrm{XH}}$ WITH RESPECT TO $\to_{\mathrm{N}\pi}$) *If $M \to_{\mathrm{XH}} N$, then $\llbracket M\rrbracket^{\mathrm{H}} a \leadsto^*_{\mathrm{N}\pi} \llbracket N\rrbracket^{\mathrm{H}} a$.*

*Proof:* By Theorem 5.9, we already know that $\llbracket M\rrbracket^{\mathrm{H}} a \leadsto^*_{\pi} \llbracket N\rrbracket^{\mathrm{H}} a$; if the new congruence rule for replication is applied, we need to check that the input name involved is reachable.

In fact, if $M$ is not in $\to_{\mathrm{XH}}$-normal form, then either:

$$
\begin{array}{ll}
(M = \lambda\vec{y}.x\overrightarrow{M}\,\overrightarrow{\langle z{:=}N\rangle}\,\langle x{:=}L\rangle): \ \llbracket\lambda\vec{y}.x\overrightarrow{M}\,\overrightarrow{\langle z{:=}N\rangle}\,\langle x{:=}L\rangle\rrbracket^{\mathrm{H}} a & \triangleq \ (\textit{proof of 5.21}) \\
\quad (\nu z_k\cdots z_1)\,((\nu y_1a_1\cdots y_na_n)\,((\nu c_1\cdots c_m)\,(x(w).\overline{c_m}\langle w\rangle \mid & \\
\qquad c_m(v,d).(!\llbracket M_m\rrbracket^{\mathrm{H}} v \mid d{\to}c_{m-1}) \mid \cdots \mid c_1(v,d).(!\llbracket M_1\rrbracket^{\mathrm{H}} v \mid d{\to}a_n)) \mid & \\
\qquad\quad \overline{a_{n-1}}\langle y_n,a_n\rangle \mid \cdots \mid \overline{a}\langle y_1,a_1\rangle) \mid !\llbracket N_1\rrbracket^{\mathrm{H}} z_1 \mid \cdots \mid !\llbracket N_k\rrbracket^{\mathrm{H}} z_k \mid !\llbracket L\rrbracket^{\mathrm{H}} x) & \equiv_{\mathrm{N}} \ (5.20) \\
\quad (\nu z_k\cdots z_1)\,((\nu y_1a_1\cdots y_na_n)\,((\nu c_1\cdots c_m)\,((\nu w)\,(\llbracket N\rrbracket^{\mathrm{H}} w \mid w(w).\overline{c_m}\langle w\rangle) \mid & \\
\qquad c_m(v,d).(!\llbracket M_m\rrbracket^{\mathrm{H}} v \mid d{\to}c_{m-1}) \mid \cdots \mid c_1(v,d).(!\llbracket M_1\rrbracket^{\mathrm{H}} v \mid d{\to}a_n)) \mid & \\
\qquad\quad \overline{a_{n-1}}\langle y_n,a_n\rangle \mid \cdots \mid \overline{a}\langle y_1,a_1\rangle) \mid !\llbracket N_1\rrbracket^{\mathrm{H}} z_1 \mid \cdots \mid !\llbracket N_k\rrbracket^{\mathrm{H}} z_k \mid !\llbracket L\rrbracket^{\mathrm{H}} x) & \leadsto_{\mathrm{R}} \\
\quad (\nu z_k\cdots z_1)\,((\nu y_1a_1\cdots y_na_n)\,((\nu c_1\cdots c_m)\,(\llbracket N\rrbracket^{\mathrm{H}} c_m \mid & \\
\qquad c_m(v,d).(!\llbracket M_m\rrbracket^{\mathrm{H}} v \mid d{\to}c_{m-1}) \mid \cdots \mid c_1(v,d).(!\llbracket M_1\rrbracket^{\mathrm{H}} v \mid d{\to}a_n)) \mid & \\
\qquad\quad \overline{a_{n-1}}\langle y_n,a_n\rangle \mid \cdots \mid \overline{a}\langle y_1,a_1\rangle) \mid !\llbracket N_1\rrbracket^{\mathrm{H}} z_1 \mid \cdots \mid !\llbracket N_k\rrbracket^{\mathrm{H}} z_k \mid !\llbracket L\rrbracket^{\mathrm{H}} x) &
\end{array}
$$

Notice that $x$ is reachable, so call by need congruence is allowed.

$$
\begin{array}{ll}
(M = \lambda\vec{y}.(\lambda x.P)Q\overrightarrow{M}\,\overrightarrow{\langle z{:=}N\rangle}): \ \llbracket\lambda\vec{y}.(\lambda x.P)Q\overrightarrow{M}\,\overrightarrow{\langle z{:=}N\rangle}\rrbracket^{\mathrm{H}} a & \triangleq \ (5.21) \\
\quad (\nu z_k\cdots z_1)\,((\nu y_1a_1\cdots y_na_n)\,((\nu c_1\cdots c_m)\,(\llbracket(\lambda x.P)Q\rrbracket^{\mathrm{H}} c_m \mid c_m(v,d).(!\llbracket M_m\rrbracket^{\mathrm{H}} v \mid d{\to}c_{m-1}) \mid \cdots \mid & \\
\qquad c_1(v,d).(!\llbracket M_1\rrbracket^{\mathrm{H}} v \mid d{\to}a_n)) \mid \overline{a_{n-1}}\langle y_n,a_n\rangle \mid \cdots \mid \overline{a}\langle y_1,a_1\rangle) \mid !\llbracket N_1\rrbracket^{\mathrm{H}} z_1 \mid \cdots \mid !\llbracket N_k\rrbracket^{\mathrm{H}} z_k) & \triangleq \\
\quad (\nu z_k\cdots z_1)\,((\nu y_1a_1\cdots y_na_n)\,((\nu c_1\cdots c_m)\,((\nu c)\,((\nu xb)\,(\llbracket P\rrbracket^{\mathrm{H}} b \mid \overline{c}\langle x,b\rangle) \mid c(v,d).(!\llbracket Q\rrbracket^{\mathrm{H}} v \mid d{\to}c_m))\mid & \\
\qquad c_m(v,d).(!\llbracket M_m\rrbracket^{\mathrm{H}} v \mid d{\to}c_{m-1}) \mid \cdots \mid c_1(v,d).(!\llbracket M_1\rrbracket^{\mathrm{H}} v \mid d{\to}a_n)) \mid & \\
\qquad\quad \overline{a_{n-1}}\langle y_n,a_n\rangle \mid \cdots \mid \overline{a}\langle y_1,a_1\rangle) \mid !\llbracket N_1\rrbracket^{\mathrm{H}} z_1 \mid \cdots \mid !\llbracket N_k\rrbracket^{\mathrm{H}} z_k) & \to_{\mathrm{N}\pi} (c) \\
\quad (\nu z_k\cdots z_1)\,((\nu y_1a_1\cdots y_na_n)\,((\nu c_1\cdots c_m)\,((\nu xb)\,(\llbracket P\rrbracket^{\mathrm{H}} b \mid !\llbracket Q\rrbracket^{\mathrm{H}} x \mid b{\to}c_m) \mid & \\
\qquad c_m(v,d).(!\llbracket M_m\rrbracket^{\mathrm{H}} v \mid d{\to}c_{m-1}) \mid \cdots \mid c_1(v,d).(!\llbracket M_1\rrbracket^{\mathrm{H}} v \mid d{\to}a_n)) \mid & \\
\qquad\quad \overline{a_{n-1}}\langle y_n,a_n\rangle \mid \cdots \mid \overline{a}\langle y_1,a_1\rangle) \mid !\llbracket N_1\rrbracket^{\mathrm{H}} z_1 \mid \cdots \mid !\llbracket N_k\rrbracket^{\mathrm{H}} z_k) & \square
\end{array}
$$

Using Lemma 5.21, we can now show our termination result with respect to call by need reduction.

**Theorem 5.23** *i ) If $M \to^*_{\mathrm{XH}} N$, with $N$ in $\to_{\mathrm{XH}}$-normal from, then $\llbracket M\rrbracket^{\mathrm{H}} a{\downarrow}_{\mathrm{N}\pi}$.*
*ii ) If $M \to^*_{\beta} N$, with $N$ in head-normal from, then $\llbracket M\rrbracket^{\mathrm{H}} a{\downarrow}_{\mathrm{N}\pi}$.*

*Proof:* *i* ) If $M \to^*_{\mathrm{XH}} N$, with $N$ in $\to_{\mathrm{XH}}$-normal from, then $N = \lambda\vec{y}.x\overrightarrow{M}\,\overrightarrow{\langle z{:=}N\rangle}$. By Theorem 5.11, $\llbracket M\rrbracket^{\mathrm{H}} a \leadsto^*_{\pi} \llbracket N\rrbracket^{\mathrm{H}} a$; also, by Lemma 5.21, $\llbracket N\rrbracket^{\mathrm{H}} a$ is in call by need normal form, so, just following the call by need reductions out of $\llbracket M\rrbracket^{\mathrm{H}} a$, we can conclude $\llbracket M\rrbracket^{\mathrm{H}} a{\downarrow}_{\mathrm{N}\pi}$.

*ii* ) If $M \to^*_{\beta} N$, with $N$ in head-normal from, then $N = \lambda\vec{x}.y\overrightarrow{N}$; then, by Proposition 3.8, there exists $P$ such that $M \to^*_{\mathrm{XH}} P$ and $P \to^*_{:=} N$; this implies that $P = \lambda\vec{x}.y\overrightarrow{P}\,\overrightarrow{\langle z{:=}L\rangle}$ and, for all $i \in \underline{m}$, $P_i\,\overrightarrow{\langle z{:=}L\rangle} \to^*_{:=} N_i$, and $y \notin \{z_1,\ldots,z_k\}$. Then, by Theorem 5.22, $\llbracket M\rrbracket^{\mathrm{H}} a \leadsto^*_{\mathrm{N}\pi} \llbracket P\rrbracket^{\mathrm{H}} a$; also, by Lemma 5.21, $\llbracket P\rrbracket^{\mathrm{H}} a$ is in call by need normal form, so, just following the reductions out of $\llbracket M\rrbracket^{\mathrm{H}} a$, we can conclude $\llbracket M\rrbracket^{\mathrm{H}} a{\downarrow}_{\mathrm{N}\pi}$. $\square$

Theorem 5.23 immediately leads to:

*Corollary 5.24 If $M{\downarrow}_{\beta}$, then $\llbracket M\rrbracket^{\mathrm{H}} a{\downarrow}_{\mathrm{N}\pi}$.*

As above, we can state the correct completeness result with respect to $\to_{\mathrm{N}\pi}$:

**Theorem 5.25** (OPERATIONAL COMPLETENESS FOR $\llbracket\cdot\rrbracket^{\mathrm{H}}\cdot$ WITH RESPECT TO $\to_{\mathrm{XH}}$) *If $\llbracket M\rrbracket^{\mathrm{H}} a \to_{\mathrm{N}\pi} Q$, then there exists $N$ such that $M \to^*_{\mathrm{XH}} N$, and $Q \to^*_{\mathrm{N}\pi} R$ and $R \leadsto^*_{\mathrm{R}}, \leadsto^*_{\mathrm{G}} \llbracket N\rrbracket^{\mathrm{H}} a$.*

*Proof:* Since, as for Theorem 5.19, the unwanted synchronisations are blocked, the proof is similar. $\square$

### 5.4 On Renaming

By Theorem 5.9, renaming might be used during the reduction. One could think that therefore our result is weaker than that of Milner, since that is stated by mapping lazy reduction onto straight $\pi$-reduction. However, in this section we will show that we can do without renaming when simulating lazy reductions for closed terms, thereby emulating Milner's result. As a consequence, it is safe to say that renaming is the price we pay for the capability to deal with reductions under abstraction, as well as that of open terms.

As an illustration of this fact, notice that, as shown in Figure 2, we *can* run the $\pi$-process $\llbracket(\lambda x.xx)(\lambda y.y)\rrbracket^{\mathrm{H}} a$ without using renaming, Notice that there we perform the two substitutions without resorting to the renaming of outputs of translated $\lambda$-terms; these are executed after the translations have participated in the execution, and take place at the end.

In the proof of Theorem 5.9, we perform a renaming (*i.e.* need the equivalence via Lemma 5.6) when dealing with the case $(\nu x)\,(x(w).\overline{c_1}\langle w\rangle \mid \llbracket N\rrbracket^{\mathrm{H}} x) \leadsto_{\mathrm{R}} \llbracket N\rrbracket^{\mathrm{H}} c_1$. Now if $N$ reduces to an abstraction $\lambda z.N'$, then (without loss of generatily)

$$\begin{array}{rcl}(\nu x)\,(x(w).\overline{c_1}\langle w\rangle \mid \llbracket N\rrbracket^{\mathrm{H}} x) & \leadsto_{\pi}^{*} & (\nu x)\,(x(w).\overline{c_1}\langle w\rangle \mid \llbracket \lambda z.N'\rrbracket^{\mathrm{H}} x) \\ & \stackrel{\Delta}{=} & (\nu x)\,(x(w).\overline{c_1}\langle w\rangle \mid (\nu zb)\,(\llbracket N'\rrbracket^{\mathrm{H}} b \mid \overline{x}\langle z,b\rangle)) \\ & \rightarrow_{\pi} & (\nu zb)\,(\llbracket N'\rrbracket^{\mathrm{H}} b \mid \overline{c_1}\langle z,b\rangle)\end{array}$$

so the renaming gets executed explicitly.

When performing a 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 the interpretation of this latter term reduces (without renaming) as:

$$\begin{array}{rcl}\llbracket(\lambda x.P)(\lambda y.Q)\rrbracket^{\mathrm{H}} a & \stackrel{\Delta}{=} & (\nu c)\,((\nu xb)\,(\llbracket P\rrbracket^{\mathrm{H}} b \mid \overline{c}\langle x,b\rangle) \mid c(v,d).\,(!\,(\nu yb)\,(\llbracket Q\rrbracket^{\mathrm{H}} b \mid \overline{v}\langle y,b\rangle) \mid d\rightarrow a)) \\ & \rightarrow_{\pi}(c) & (\nu xb')\,(\llbracket P\rrbracket^{\mathrm{H}} b' \mid !\,(\nu yb)\,(\llbracket Q\rrbracket^{\mathrm{H}} b \mid \overline{x}\langle y,b\rangle) \mid b'\rightarrow a)\end{array}$$

Now assume $P = x\vec{P}$ (only then does $P\,\langle x{:=}Q\rangle$ reduce), then, since $\lambda x.x\vec{P}$ is closed, each $P_i$ is either $x$ or an abstraction. Then this reduction continues as follows:

$$\begin{array}{ll}(\nu xb')\,(\llbracket x\vec{P}\rrbracket^{\mathrm{H}} b' \mid !\llbracket \lambda y.Q\rrbracket^{\mathrm{H}} x \mid b'\rightarrow a) & \stackrel{\Delta}{=} \\ (\nu xb_1)\,((\nu c_n)\,(\llbracket xP_1\cdots P_{n-1}\rrbracket^{\mathrm{H}} c_n \mid c_n(v,d).\,(!\llbracket P_n\rrbracket^{\mathrm{H}} v \mid d\rightarrow b')) \mid !\llbracket \lambda y.Q\rrbracket^{\mathrm{H}} x \mid b'\rightarrow a) & \stackrel{\Delta}{=} \\ (\nu xb')\,((\nu c_n)\,(\cdots(\nu c_1)\,(x(w).\overline{c_1}\langle w\rangle \mid c_1(v,d).\,(!\llbracket P_1\rrbracket^{\mathrm{H}} v \mid d\rightarrow c_2)) \mid \cdots \mid & \\ \qquad c_n(v,d).\,(!\llbracket P_n\rrbracket^{\mathrm{H}} v \mid d\rightarrow b')) \mid !\llbracket \lambda y.Q\rrbracket^{\mathrm{H}} x \mid b'\rightarrow a) & \equiv,\stackrel{\Delta}{=} \\ (\nu xb')\,((\nu c_n)\,(\cdots(\nu c_1)\,(x(w).\overline{c_1}\langle w\rangle \mid c_1(v,d).\,(!\llbracket P_1\rrbracket^{\mathrm{H}} v \mid d\rightarrow c_2)) \mid \cdots \mid & \\ \qquad c_n(v,d).\,(!\llbracket P_n\rrbracket^{\mathrm{H}} v \mid d\rightarrow b')) \mid (\nu yb)\,(\llbracket Q\rrbracket^{\mathrm{H}} b \mid \overline{x}\langle y,b\rangle) \mid !\llbracket \lambda y.Q\rrbracket^{\mathrm{H}} x \mid b'\rightarrow a) & \rightarrow_{\pi}(x) \\ (\nu xb')\,((\nu c_n)\,(\cdots(\nu c_1)\,((\nu zb)\,(\llbracket Q[z/y]\rrbracket^{\mathrm{H}} b \mid \overline{c_1}\langle z,b\rangle) \mid c_1(v,d).\,(!\llbracket P_1\rrbracket^{\mathrm{H}} v \mid d\rightarrow c_2)) \mid \cdots \mid & \\ \qquad c_n(v,d).\,(!\llbracket P_n\rrbracket^{\mathrm{H}} v \mid d\rightarrow b')) \mid !\llbracket \lambda y.Q\rrbracket^{\mathrm{H}} x \mid b'\rightarrow a) & \stackrel{\Delta}{=} \\ (\nu xb')\,(\llbracket(\lambda z.Q[z/y])\vec{P}\rrbracket^{\mathrm{H}} b' \mid !\llbracket \lambda y.Q\rrbracket^{\mathrm{H}} x \mid b'\rightarrow a) & \end{array}$$

without renaming (notice the $\alpha$-conversion). Notice that, if $P_1$ is an abstraction, $(\lambda z.Q[z/y])P_1$ has the structure of the term we started our reasoning with.

Otherwise, $P_1 = x$ and we get:

$$\begin{array}{ll}(\nu xb')\,(\llbracket(\lambda z.Q[z/y])xP_2\cdots P_n\rrbracket^{\mathrm{H}} b' \mid !\llbracket \lambda y.Q\rrbracket^{\mathrm{H}} x \mid b'\rightarrow a) & \stackrel{\Delta}{=},\equiv \\ (\nu xb')\,((\nu c_n\cdots\nu c_2)\,((\nu c_1)\,((\nu zb)\,(\llbracket Q[z/y]\rrbracket^{\mathrm{H}} b \mid \overline{c_1}\langle z,b\rangle) \mid c_1(v,d).\,(!\llbracket x\rrbracket^{\mathrm{H}} v \mid d\rightarrow c_2)) \mid & \\ \qquad c_2(b,d).\,(!\llbracket P_2\rrbracket^{\mathrm{H}} b \mid d\rightarrow c_3) \mid \cdots \mid c_n(v,d).\,(!\llbracket P_n\rrbracket^{\mathrm{H}} v \mid d\rightarrow b')) \mid !\llbracket \lambda y.Q\rrbracket^{\mathrm{H}} x \mid b'\rightarrow a) & \rightarrow_{\pi}(c_1) \\ (\nu xb')\,((\nu c_n\cdots\nu c_2)\,((\nu zb)\,(\llbracket Q[z/y]\rrbracket^{\mathrm{H}} b \mid x(w).\overline{z}\langle w\rangle \mid !\llbracket x\rrbracket^{\mathrm{H}} z \mid b\rightarrow c_2) \mid & \\ \qquad c_2(b,d).\,(!\llbracket P_2\rrbracket^{\mathrm{H}} b \mid d\rightarrow c_3) \mid \cdots \mid c_n(v,d).\,(!\llbracket P_n\rrbracket^{\mathrm{H}} v \mid d\rightarrow b')) \mid !\llbracket \lambda y.Q\rrbracket^{\mathrm{H}} x \mid b'\rightarrow a) & \end{array}$$

$$
\begin{array}{ll}
\llbracket (\lambda xy.x(\lambda z.z))(\lambda p.p) \rrbracket^{\mathbf{H}} a & \stackrel{\Delta}{=} \\
(\nu c)\,((\nu xb)\,(\llbracket \lambda y.x(\lambda z.z) \rrbracket^{\mathbf{H}} b \mid \overline{c}\langle x,b\rangle) \mid c(v,d).\,(!\llbracket \lambda p.p \rrbracket^{\mathbf{H}} v \mid d \rightarrow a)) & \rightarrow_{\pi} (c) \\
(\nu xb)\,(\llbracket \lambda y.x(\lambda z.z) \rrbracket^{\mathbf{H}} b \mid !\llbracket \lambda p.p \rrbracket^{\mathbf{H}} x \mid b \rightarrow a) & \stackrel{\Delta}{=} \\
(\nu xb)\,((\nu yb_1)\,((\nu c)\,(x(w).\,\overline{c}\langle w\rangle \mid c(v,d).\,(!\llbracket \lambda z.z \rrbracket^{\mathbf{H}} v \mid d \rightarrow b_1)) \mid \overline{b}\langle y,b_1\rangle) \mid !\,(\nu pb)\,(\llbracket p \rrbracket^{\mathbf{H}} b \mid \overline{x}\langle p,b\rangle) \mid b \rightarrow a) & \rightarrow_{\pi} (x) \\
(\nu yb_1)\,((\nu xb)\,((\nu c)\,(\llbracket \lambda p.p \rrbracket^{\mathbf{H}} c \mid c(v,d).\,(!\llbracket \lambda z.z \rrbracket^{\mathbf{H}} v \mid d \rightarrow b)) \mid !\llbracket \lambda p.p \rrbracket^{\mathbf{H}} x \mid b \rightarrow b_1) \mid \overline{a}\langle y,b_1\rangle) & \stackrel{\Delta}{=} \\
(\nu yb_1)\,((\nu xb)\,((\nu c)\,((\nu pb)\,(\llbracket p \rrbracket^{\mathbf{H}} b \mid \overline{c}\langle p,b\rangle) \mid c(v,d).\,(!\llbracket \lambda z.z \rrbracket^{\mathbf{H}} v \mid d \rightarrow b)) \mid !\llbracket \lambda p.p \rrbracket^{\mathbf{H}} x \mid b \rightarrow b_1) \mid \overline{a}\langle y,b_1\rangle) & \rightarrow_{\pi} (c) \\
(\nu yb_1)\,((\nu xb)\,((\nu pb_2)\,(\llbracket p \rrbracket^{\mathbf{H}} b_2 \mid !\llbracket \lambda z.z \rrbracket^{\mathbf{H}} p \mid b_2 \rightarrow b) \mid !\llbracket \lambda p.p \rrbracket^{\mathbf{H}} x \mid b \rightarrow b_1) \mid \overline{a}\langle y,b_1\rangle) & \stackrel{\Delta}{=} \\
(\nu yb_1)\,((\nu xb)\,((\nu pb_2)\,(p(w).\,\overline{b_2}\langle w\rangle \mid !\,(\nu zb)\,(\llbracket z \rrbracket^{\mathbf{H}} b \mid \overline{p}\langle z,b\rangle) \mid b_2 \rightarrow b) \mid !\llbracket \lambda p.p \rrbracket^{\mathbf{H}} x \mid b \rightarrow b_1) \mid \overline{a}\langle y,b_1\rangle) & \rightarrow_{\pi} (p) \\
(\nu yb_1)\,((\nu xb)\,((\nu pb_2)\,((\nu zb_3)\,(\llbracket z \rrbracket^{\mathbf{H}} b_3 \mid \overline{b_2}\langle z,b_3\rangle) \mid !\llbracket \lambda z.z \rrbracket^{\mathbf{H}} p \mid b_2 \rightarrow b) \mid !\llbracket \lambda p.p \rrbracket^{\mathbf{H}} x \mid b \rightarrow b_1) \mid \overline{a}\langle y,b_1\rangle) & \stackrel{\Delta}{=} \\
(\nu yb_1)\,((\nu xb)\,((\nu pb_2)\,((\nu zb_3)\,(\llbracket z \rrbracket^{\mathbf{H}} b_3 \mid \overline{b_2}\langle z,b_3\rangle) \mid !\llbracket \lambda z.z \rrbracket^{\mathbf{H}} p \mid b_2 \rightarrow b) \mid !\llbracket \lambda p.p \rrbracket^{\mathbf{H}} x \mid b \rightarrow b_1) \mid \overline{a}\langle y,b_1\rangle) & \rightarrow_{\pi} (b_2,b) \\
(\nu yb_1)\,((\nu x)\,((\nu p)\,((\nu zb_3)\,(\llbracket z \rrbracket^{\mathbf{H}} b_3 \mid \overline{b_1}\langle z,b_3\rangle) \mid !\llbracket \lambda z.z \rrbracket^{\mathbf{H}} p) \mid !\llbracket \lambda p.p \rrbracket^{\mathbf{H}} x) \mid \overline{a}\langle y,b_1\rangle) & \equiv, \stackrel{\Delta}{=} \\
(\nu x)\,((\nu p)\,((\nu yb_1)\,(\llbracket \lambda z.z \rrbracket^{\mathbf{H}} b_1 \mid \overline{a}\langle y,b_1\rangle) \mid !\llbracket \lambda z.z \rrbracket^{\mathbf{H}} p) \mid !\llbracket \lambda p.p \rrbracket^{\mathbf{H}} x) & \stackrel{\Delta}{=} \\
(\nu x)\,((\nu p)\,(\llbracket \lambda yz.z \rrbracket^{\mathbf{H}} a \mid !\llbracket \lambda z.z \rrbracket^{\mathbf{H}} p) \mid !\llbracket \lambda p.p \rrbracket^{\mathbf{H}} x) & \stackrel{\Delta}{=} \\
\llbracket \lambda yz.z\,\langle p := \lambda z.z\rangle\,\langle x := \lambda p.p\rangle \rrbracket^{\mathbf{H}} a & \equiv \\
\llbracket \lambda yz.z \rrbracket^{\mathbf{H}} a \mid (\nu xp)\,(\llbracket \lambda z.z \rrbracket^{\mathbf{H}} p \mid \llbracket \lambda p.p \rrbracket^{\mathbf{H}} x) & \sim_{\mathrm{G}} \\
\llbracket \lambda yz.z \rrbracket^{\mathbf{H}} a &
\end{array}
$$

Figure 5: Running $\llbracket (\lambda xy.x(\lambda z.z))(\lambda p.p) \rrbracket^{\mathbf{H}} a$ to normal form without renaming.

Now assume $Q[z/y]$ is of the shape $zQ_1 \cdots Q_m$, then $\llbracket Q[z/y] \rrbracket^{\mathbf{H}} b = \llbracket zQ_1 \cdots Q_m \rrbracket^{\mathbf{H}} b$ so the above process is of the shape

$$
\begin{array}{l}
(\nu xb')\,((\nu c_n \cdots \nu c_2)\,((\nu yb)\,((\nu c'_m \cdots \nu c'_1)\,(z(w).\,\overline{c_1}\langle w\rangle \mid \\
\quad c'_1(b,d).\,(!\llbracket Q_1 \rrbracket^{\mathbf{H}} b \mid d \rightarrow c'_2) \mid \cdots \mid c'_m(v,d).\,(!\llbracket P_n \rrbracket^{\mathbf{H}} v \mid d \rightarrow b)) \mid \\
\quad\quad x(w).\,\overline{z}\langle w\rangle \mid !\llbracket x \rrbracket^{\mathbf{H}} z \mid b \rightarrow c_2) \mid \\
\quad\quad\quad c_2(b,d).\,(!\llbracket P_2 \rrbracket^{\mathbf{H}} b \mid d \rightarrow c_3)) \mid \cdots \mid c_n(v,d).\,(!\llbracket P_n \rrbracket^{\mathbf{H}} v \mid d \rightarrow b') \mid !\llbracket \lambda y.Q \rrbracket^{\mathbf{H}} x \mid b' \rightarrow a)
\end{array}
$$

which reduces over $x$ and $z$ to:

$$
\begin{array}{l}
(\nu xb')\,((\nu c_n \cdots \nu c_2)\,((\nu yb)\,((\nu c'_m \cdots \nu c'_1)\,(\llbracket \lambda v.Q[v/y] \rrbracket^{\mathbf{H}} c_1 \mid \\
\quad c'_1(b,d).\,(!\llbracket Q_1 \rrbracket^{\mathbf{H}} b \mid d \rightarrow c'_2) \mid \cdots \mid c'_m(v,d).\,(!\llbracket P_n \rrbracket^{\mathbf{H}} v \mid d \rightarrow b)) \mid \\
\quad\quad !\llbracket x \rrbracket^{\mathbf{H}} z \mid b \rightarrow c_2) \mid \\
\quad\quad\quad c_2(b,d).\,(!\llbracket P_2 \rrbracket^{\mathbf{H}} b \mid d \rightarrow c_3)) \mid \cdots \mid c_n(v,d).\,(!\llbracket P_n \rrbracket^{\mathbf{H}} v \mid d \rightarrow b') \mid !\llbracket \lambda y.Q \rrbracket^{\mathbf{H}} x \mid b' \rightarrow a)
\end{array}
$$

without renaming.

So, when simulating lazy explicit reduction on closed terms, renaming can be postponed, and the relation $\sim_{\mathrm{R}}$ is not needed. This result clearly shows that renaming is only essential to model reductions that are not lazy; and even for those, when modelling the reduction of the redex $(\lambda x.P)C$ using $\rightarrow_{\mathbf{xH}}$, where $C$ is a closed term, renaming is not needed.

*Example 5.26* Consider the garbage-collection-free explicit head reduction

$$
\begin{array}{rcl}
(\lambda xy.x(\lambda z.z))(\lambda p.p) & \rightarrow^{\not c}_{\mathbf{xH}} & \lambda y.x(\lambda z.z)\,\langle x := \lambda p.p\rangle \\
& \rightarrow^{\not c}_{\mathbf{xH}} & \lambda y.x(\lambda z.z)\,\langle x := \lambda p.p\rangle \\
& \rightarrow^{\not c}_{\mathbf{xH}} & \lambda y.(\lambda p.p)(\lambda z.z)\,\langle x := \lambda p.p\rangle \\
& \rightarrow^{\not c}_{\mathbf{xH}} & \lambda y.p\,\langle p := \lambda z.z\rangle\,\langle x := \lambda p.p\rangle \\
& \rightarrow^{\not c}_{\mathbf{xH}} & \lambda y.\lambda z.z\,\langle p := \lambda z.z\rangle\,\langle x := \lambda p.p\rangle
\end{array}
$$

We can simulate this, without renaming, using our encoding $\llbracket \cdot \rrbracket^{\mathbf{H}} \cdot$, as shown in Figure 5. Notice that, using explicit lazy reduction, we can only perform:

$$
(\lambda xy.x(\lambda z.z))(\lambda p.p) \;\rightarrow^{\not c}_{\mathbf{xL}}\; \lambda y.x(\lambda z.z)\,\langle x := \lambda p.p\rangle
$$

Summarising, as in Corollary 4.7 we can now state:

*Corollary 5.27* *i ) If $M$ is closed, and $M \to^*_{\mathbf{XL}} (\lambda y.N)\overrightarrow{\langle x := L\rangle}$, then $\llbracket M \rrbracket^{\mathbf{H}} a \to^*_{\pi'} \sim_{\mathrm{G}} (\nu\vec{x})\,(\llbracket \lambda y.N \rrbracket^{\mathbf{H}} a \mid !\overrightarrow{\llbracket L \rrbracket^{\mathbf{H}} x})$.*

*ii ) If $M$ is closed, and $M \to^{\not{c}*}_{\mathbf{XL}} (\lambda y.N)\overrightarrow{\langle x := L\rangle}$, then $\llbracket M \rrbracket^{\mathbf{H}} a \to^*_{\pi} (\nu\vec{x})\,(\llbracket \lambda y.N \rrbracket^{\mathbf{H}} a \mid !\overrightarrow{\llbracket L \rrbracket^{\mathbf{H}} x})$.*

*iii ) If $M$ is closed and $M \to^*_{\mathrm{L}} (\lambda y.N)$, then there exists $N', \vec{x}, \vec{L}$ such that $\llbracket M \rrbracket^{\mathbf{H}} a \to^*_{\pi} (\overline{\nu x})\,(\llbracket \lambda y.N' \rrbracket^{\mathbf{H}} a \mid !\overrightarrow{\llbracket L \rrbracket^{\mathbf{H}} x})$, and $\lambda y.N'\overrightarrow{\langle x := L\rangle} \to^*_{:=} \lambda x.N$.*

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

# 6 Context assignment

The $\pi$-calculus is equipped with a rich type theory [32]: 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 [28], to sophisticated linear types in [22], which studies a relation between Call-by-Value $\lambda\mu$ and a linear $\pi$-calculus. There linearisation is used to be able to achieve processes that are functions, by allowing output over one channel name only, in a ($\lambda$-calculus) natural deduction style. Moreover, the translation presented in [22] is type dependent, in that, for each term, different $\pi$-processes are assigned, depending on the original type; this makes the translation quite cumbersome.

The notion of context assignment for processes in $\pi$ we define in this section was first presented in [5] and differs quite drastically from the standard type system presented in [32]. 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 $\pi$-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 [5] to establish preservation of assignable types under the interpretation of the sequent calculus $\mathcal{X}$ into the $\pi$-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 [17], 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 6.1** (Types and Contexts) *i* ) The set of types is defined by the grammar:

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

where $\varphi$ is a basic type of which there are infinitely many. The types considered in this paper are normally known as *simple* (or *Curry*) types.

*ii* ) An *input context* $\Gamma$ 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 $\Gamma_1, \Gamma_2$ to mean the *compatible union* of $\Gamma_1$ and $\Gamma_2$ (if $\Gamma_1$ contains $n{:}A_1$ and $\Gamma_2$ contains $n{:}A_2$, then $A_1 = A_2$), and write $\Gamma, n{:}A$ for $\Gamma, \{n{:}A\}$.

*iii* ) *Output* contexts $\Delta$, and the notions $\Delta_1, \Delta_2$, and $n{:}A, \Delta$ are defined in a similar way.

*iv* ) 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 $\Gamma$ is not defined on $n$.

**Definition 6.2** ((Classical) Context Assignment) Context assignment for the $\pi$-calculus with pairing is defined by the following sequent system:

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

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

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

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

We write $P : \Gamma \vdash_\pi \Delta$ if there exists a derivation using these rules that has this expression in the conclusion and demand that, if $n{:}A \in \Gamma$ and $n{:}B \in \Delta$, then $A = B$; we 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 $\overline{a}\langle a\rangle : \vdash_\pi a{:}A$.

*Example 6.3* Although we have no rule $(\textit{pair-in})$, it is admissible, since we can derive

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

so the following rule is admissible:

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

Notice that this notion of type assignment does not (directly) relate back to any known logical calculus. For example, rules $(|)$ and $(!)$ do not change the contexts; moreover, rule $(\nu)$ just removes a formula.

The presence of *weakening* 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

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

so switching, without any scruples, to multiplicative style, whenever convenient.

We have a soundness (witness reduction) result for our notion of type assignment for $\pi$ as shown in [5].

**Theorem 6.4** (Witness reduction [5]) *If $P : \Gamma \vdash_\pi \Delta$ and $P \to_\pi^* Q$, then $Q : \Gamma \vdash_\pi \Delta$.*

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

**Definition 6.5** (Curry type assignment for $\lambda\mathbf{x}$) Curry type assignment for $\lambda\mathbf{x}$ is defined

through the following natural deduction rules:

$$(Ax):\ \frac{}{\Gamma, x{:}A \vdash_\lambda x{:}A} \qquad (cut):\ \frac{\Gamma, x{:}A \vdash_\lambda M{:}B \quad \Gamma \vdash_\lambda N{:}A}{\Gamma \vdash_\lambda M\langle x{:=}N\rangle{:}B}$$

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

We write $\Gamma \vdash_\lambda M{:}A$ if there exists a derivation using these rules that has this expression in the conclusion.

We can now show that typeability is preserved by $\llbracket \cdot \rrbracket^{\mathbf{H}} \cdot$:

**Theorem 6.6** *If $\Gamma \vdash_\lambda M{:}A$, then $\llbracket M \rrbracket^{\mathbf{H}} a : \Gamma \vdash_\pi a{:}A$.*

*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 $\llbracket x \rrbracket^{\mathbf{H}} a = x(w).\overline{a}\langle w\rangle$, and that

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

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

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

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

$(\to E)$ : 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 $\mathcal{D}_1 :: \llbracket P \rrbracket^{\mathbf{H}} c : \Gamma \vdash_\pi c{:}B\to A$ and $\mathcal{D}_2 :: \llbracket Q \rrbracket^{\mathbf{H}} b : \Gamma \vdash_\pi b{:}B$, and we can construct:

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

and $(\nu c)(\llbracket P \rrbracket^{\mathbf{H}} c \mid c(v,d).(!\llbracket Q \rrbracket^{\mathbf{H}} v \mid d{\to}a)) = \llbracket PQ \rrbracket^{\mathbf{H}} a$.

$(cut)$ : Then $M = P\langle x{:=}Q\rangle$, and there exists $B$ such that $\Gamma, x{:}B \vdash_\lambda P{:}A$ and $\Gamma \vdash_\lambda Q{:}B$, and $\llbracket P\langle x{:=}Q\rangle \rrbracket^{\mathbf{F}} a = (\nu x)(\llbracket P \rrbracket^{\mathbf{F}} a \mid !\llbracket Q \rrbracket^{\mathbf{F}} x)$. By induction, there exist $\mathcal{D}_1 :: \llbracket P \rrbracket^{\mathbf{F}} c : \Gamma, x{:}B \vdash_\pi a{:}A$ and $\mathcal{D}_2 :: \llbracket Q \rrbracket^{\mathbf{F}} x : \Gamma \vdash_\pi x{:}B$, and we can construct:

$$\dfrac{\dfrac{\dfrac{\boxed{\mathcal{D}_1}}{\llbracket P \rrbracket^{\mathbf{H}} a : \Gamma, x{:}B \vdash_\pi a{:}A} \qquad \dfrac{\dfrac{\boxed{\mathcal{D}_2}}{\llbracket Q \rrbracket^{\mathbf{H}} x : \Gamma \vdash_\pi x{:}B}}{!\llbracket Q \rrbracket^{\mathbf{H}} x : \Gamma \vdash_\pi x{:}B}\,(!)}{\llbracket P \rrbracket^{\mathbf{H}} a \mid !\llbracket Q \rrbracket^{\mathbf{H}} x : \Gamma, x{:}B \vdash_\pi x{:}B, a{:}A}\,(|)}{(\nu x)(\llbracket P \rrbracket^{\mathbf{F}} a \mid !\llbracket Q \rrbracket^{\mathbf{F}} x) : \Gamma \vdash_\pi a{:}A}\,(\nu)$$

and $(\nu x)\,(\llbracket P \rrbracket^{\mathrm{F}}_{\mathrm{J}} a \mid !\llbracket Q \rrbracket^{\mathrm{F}}_{\mathrm{J}} x) = \llbracket P\langle x := Q\rangle \rrbracket^{\mathrm{F}}_{\mathrm{J}} a$. $\square$

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 $\lambda$-calculus, an intuitionistic natural deduction system – we need the classical, multi-conclusion character of our type assignment system for $\pi$ to achieve this result for the case $(\to E)$.

A natural question to ask is if also the processes created by Milner's translation are typeable in $\vdash_\pi$. To investigate this question, essentially following Definition 4.1, we first define an input-based translation of $\lambda$-terms - a variant of Milner's translation - into the asynchronous $\pi$-calculus with pairing by:

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

(Notice the similarity with the uniform translation $\llbracket \cdot \rrbracket^{\mathrm{U}}_{\mathrm{J}} \cdot$ from Definition **??**.) Notice that this interpretation is well behaved: using $\llbracket \cdot \rrbracket^{\mathrm{P}}_{\mathrm{J}} \cdot$, the interpretation of a $\beta$-redex reduces as follows:

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

exactly as expected.

Notice that, in this translation, all $\lambda$-calculus input variables are interpreted as output channels, and that $a$ is the only input variable for each translated $\lambda$-term; so we could hope to show " *If* $\Gamma \vdash_\lambda M : A$, *then* $\llbracket M \rrbracket^{\mathrm{P}}_{\mathrm{J}} a : a{:}A \vdash_\pi \Gamma$ ". 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 both $\llbracket P \rrbracket^{\mathrm{P}}_{\mathrm{J}} c : c{:}B \to A \vdash_\pi \Gamma$ and $\llbracket Q \rrbracket^{\mathrm{P}}_{\mathrm{J}} w : w{:}B \vdash_\pi \Gamma$, and we can construct:

$$\dfrac{\dfrac{\dfrac{\boxed{\phantom{xxxxxx}}}{\llbracket P \rrbracket^{\mathrm{P}}_{\mathrm{J}} c : c{:}B \to A \vdash_\pi \Gamma} \quad \dfrac{\dfrac{\dfrac{}{\overline{c}\langle z,a\rangle : z{:}B \vdash_\pi c{:}B \to A, a{:}A, \Gamma}\,(\textit{pair-out}) \quad \dfrac{\dfrac{\dfrac{\boxed{\phantom{xxxxxx}}}{\llbracket Q \rrbracket^{\mathrm{P}}_{\mathrm{J}} w : w{:}B \vdash_\pi \Gamma}}{z(w).\,\llbracket Q \rrbracket^{\mathrm{P}}_{\mathrm{J}} w : z{:}B \vdash_\pi \Gamma}\,(in)}{!z(w).\,\llbracket Q \rrbracket^{\mathrm{P}}_{\mathrm{J}} z : z{:}B \vdash_\pi \Gamma}\,(!)}{\overline{c}\langle z,a\rangle \mid !z(w).\,\llbracket Q \rrbracket^{\mathrm{P}}_{\mathrm{J}} z : z{:}B \vdash_\pi c{:}B \to A, a{:}A, \Gamma}\,(|)}{(\nu z)\,(\overline{c}\langle z,a\rangle \mid !z(w).\,\llbracket Q \rrbracket^{\mathrm{P}}_{\mathrm{J}} z) : \vdash_\pi c{:}B \to A, a{:}A, \Gamma}\,(\nu)}{\llbracket P \rrbracket^{\mathrm{M}}_{\mathrm{J}} c \mid (\nu z)\,(\overline{c}\langle z,a\rangle \mid !z(w).\,\llbracket Q \rrbracket^{\mathrm{P}}_{\mathrm{J}} z) : c{:}B \to A \vdash_\pi c{:}B \to A, a{:}A, \Gamma}\,(|)}{(\nu c)\,(\llbracket P \rrbracket^{\mathrm{P}}_{\mathrm{J}} c \mid (\nu z)\,(\overline{c}\langle z,a\rangle.\llbracket z := Q \rrbracket^{\mathrm{P}}_{\mathrm{J}})) : \vdash_\pi a{:}A, \Gamma}\,(\nu)$$

rather than a derivation for $(\nu c)\,(\llbracket P \rrbracket^{\mathrm{P}}_{\mathrm{J}} c \mid (\nu z)\,(\overline{c}\langle z,a\rangle.\llbracket z := Q \rrbracket^{\mathrm{P}}_{\mathrm{J}})) : a{:}A \vdash_\pi \Gamma$, as was desired. Notice that the rule (*pair-out*) 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 translation for the same reason.

## 7 A logical translation for $\to_\beta$ and $=_\beta$

In this section we will define our full translation $\llbracket \cdot \rrbracket^{\mathrm{F}}_{\mathrm{J}} \cdot$ of $\lambda\mathbf{x}$ (and thereby of the $\lambda$-calculus) into the $\pi$-calculus. We will show that, using this translation, we can fully represent $\beta$-

reduction and equality modulo contextual equivalence. This translation, where as before we name the output of terms, is defined by:

**Definition 7.1** (Full logical translation)

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

Notice that the main difference with respect to $\llbracket \cdot \rrbracket^{\mathrm{H}}_{\cdot}$ is the synchronisation in the interpretation of the application; where $\llbracket MN \rrbracket^{\mathrm{F}}_{a}$ contains $c(v,d).\,(!\,b\rightarrow v \mid d\rightarrow a) \mid !\llbracket N \rrbracket^{\mathrm{F}}_{b}$, under our first interpretation $\llbracket MN \rrbracket^{\mathrm{H}}_{a}$ has $c(v,d).\,(!\llbracket N \rrbracket^{\mathrm{F}}_{v} \mid d\rightarrow a)$, where the received input name is used directly for the interpretation of $N$, which gets placed under *input*. In $\llbracket \cdot \rrbracket^{\mathrm{F}}_{\cdot}$, the replicated term in the interpretation of the substitution is again not guarded, so can run on its own; this is intentional: we aim to interpret $\lambda\mathbf{x}$, where substitution is explicit, and reductions are allowed to take place inside $N$ in $M\langle x{:=}N\rangle$. Moreover, we will allow substitution terms to interact freely, so will not use call by need reduction.

In particular:

- for an application $MN$, the output of $M$, transmitted over $c$, is received as a pair $\langle v,d\rangle$ of input-output names in the *synchronisation cell* $c(v,d).\,(!\,b\rightarrow v \mid d\rightarrow a)$; the received input $v$ name is used to redirect the output for $N$ arriving over $b$ (since $\llbracket N \rrbracket^{\mathrm{F}}_{b}$ gets replicated, so does $b\rightarrow v$), enabling the simulation of substitution, and the received output name $d$ gets redirected to the output of the application $a$.
- since we aim to represent *all* reductions taking place inside an application $MN$, we need to express $\llbracket MN \rrbracket^{\mathrm{F}}_{a}$ in terms of $\llbracket M \rrbracket^{\mathrm{F}}_{b}$ and $\llbracket N \rrbracket^{\mathrm{F}}_{c}$, where neither can appear under an *input*, since that would imply that reduction would be blocked, as is the case for the traditional approaches.
- Our translation generates a highly parallel implementation of $\lambda$-terms, with no nesting at all; the processes we generate are, essentially, a flat parallel composition of components like
$$x(w).\,\overline{a}\langle w\rangle \qquad b(v,d).\,(!\,a\rightarrow v \mid d\rightarrow c) \qquad \overline{a}\langle y,b\rangle$$
using replication where needed. Moreover, we model explicit substitution via the communication of two processes, so can faithfully take into account all steps of $\lambda\mathbf{x}$-reduction. As a consequence, the *Substitution Lemma*, an important property in the $\lambda$-calculus, is naturally preserved by our translation.
- As for $\llbracket \cdot \rrbracket^{\mathrm{H}}_{\cdot}$, $\llbracket M \rrbracket^{\mathrm{F}}_{a}$ is a process that has only one free output, $a$.

*Example 7.2* The interpretation of a redex reduces as:

$$\begin{array}{rcl} \llbracket (\lambda x.P)Q \rrbracket^{\mathrm{F}}_{a} & \triangleq & (\nu cb)\,((\nu xb_1)\,(\llbracket P \rrbracket^{\mathrm{F}}_{b_1} \mid \overline{c}\langle x,b_1\rangle) \mid c(v,d).\,(!\,b\rightarrow v \mid d\rightarrow a) \mid !\llbracket Q \rrbracket^{\mathrm{F}}_{b}) \\ & \rightarrow_{\pi}(c) & (\nu bxb_1)\,(\llbracket P \rrbracket^{\mathrm{F}}_{b_1} \mid !\,b\rightarrow x \mid b_1\rightarrow a \mid !\llbracket Q \rrbracket^{\mathrm{F}}_{b}) \end{array}$$

Since reduction in $\lambda\mathbf{x}$ implements $\beta$-reduction by at least performing this step, this implies that also with $\llbracket \cdot \rrbracket^{\mathrm{F}}_{\cdot}$ we model each $\beta$-reduction step by at least one $\pi$-reduction.

Even though our translation $\llbracket \cdot \rrbracket^{\mathrm{F}}_{\cdot}$ is fundamentally different form the one we presented in [9], we can still show that typeability is preserved:

**Theorem 7.3** ($\llbracket \cdot \rrbracket^{\mathrm{F}}_{\cdot}$ preserves assignable types) *If $\Gamma \vdash_{\lambda} M{:}A$, then $\llbracket M \rrbracket^{\mathrm{F}}_{a} : \Gamma \vdash_{\pi} a{:}A$.*

*Proof:* By induction on the structure of derivations in $\vdash_\lambda$; we only show the one interesting case.

$(\to E)$ : Then $M = PQ$, and there exists $B$ such that $\Gamma \vdash_\lambda P : B \to A$ and $\Gamma \vdash_\lambda Q : B$, and $\llbracket PQ \rrbracket^{\mathsf{F}}_{\rrbracket} a = (\nu cb)\,(\llbracket P \rrbracket^{\mathsf{F}}_{\rrbracket} c \mid c(v,d).\,(!\,b \twoheadrightarrow v \mid d \twoheadrightarrow a) \mid !\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} b)$. By induction, there exist $\mathcal{D}_1 :: \llbracket P \rrbracket^{\mathsf{F}}_{\rrbracket} c : \Gamma \vdash_\pi c{:}B \to A$ and $\mathcal{D}_2 :: \llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} b : \Gamma \vdash_\pi b{:}B$, and we can construct:

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{\mathcal{D}_1}{\llbracket P \rrbracket^{\mathsf{F}}_{\rrbracket} c : \Gamma \vdash_\pi c{:}B \to A}
\quad
\dfrac{
\dfrac{
\dfrac{
\dfrac{\overline{v}\langle w\rangle : \Gamma, w{:}B \vdash_\pi v{:}B, w{:}B, \Delta}{b \twoheadrightarrow v : b{:}B \vdash_\pi v{:}B}\,(out)\,(in)
}{!\,b \twoheadrightarrow v : b{:}B \vdash_\pi v{:}B}\,(!)
\quad
\dfrac{\overline{a}\langle w\rangle : \Gamma, w{:}A \vdash_\pi a{:}A, w{:}A, \Delta}{d \twoheadrightarrow a : d{:}A \vdash_\pi a{:}A}\,(out)\,(in)
}{!\,b \twoheadrightarrow v \mid d \twoheadrightarrow a : \Gamma, b{:}B, d{:}A \vdash_\pi v{:}B, a{:}A}\,(|)
}{c(v,d).\,(!\,b \twoheadrightarrow v \mid d \twoheadrightarrow a) : \Gamma, b{:}B, c{:}B \to A \vdash_\pi a{:}A}\,(pair\text{-}in)
\quad
\dfrac{\dfrac{\mathcal{D}_2}{\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} b : \Gamma \vdash_\pi b{:}B}}{!\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} b : \Gamma \vdash_\pi b{:}B}\,(!)
}{\llbracket P \rrbracket^{\mathsf{F}}_{\rrbracket} c \mid c(v,d).\,(!\,b \twoheadrightarrow v \mid d \twoheadrightarrow a) \mid !\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} b : \Gamma, b{:}B, c{:}B \to A \vdash_\pi c{:}B \to A, a{:}A, b{:}B}\,(|)
}{(\nu b)\,(\llbracket P \rrbracket^{\mathsf{F}}_{\rrbracket} c \mid c(v,d).\,(!\,b \twoheadrightarrow v \mid d \twoheadrightarrow a) \mid !\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} b) : \Gamma, c{:}B \to A \vdash_\pi c{:}B \to A, a{:}A}\,(\nu)
}{(\nu cb)\,(\llbracket P \rrbracket^{\mathsf{F}}_{\rrbracket} c \mid c(v,d).\,(!\,b \twoheadrightarrow v \mid d \twoheadrightarrow a) \mid !\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} b) : \Gamma \vdash_\pi a{:}A}\,(\nu)
$$

and $(\nu cb)\,(\llbracket P \rrbracket^{\mathsf{F}}_{\rrbracket} c \mid c(v,d).\,(!\,b \twoheadrightarrow v \mid d \twoheadrightarrow a) \mid !\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} b) = \llbracket PQ \rrbracket^{\mathsf{F}}_{\rrbracket} a$. $\square$

We will below show that our translation fully respects the explicit reduction $\to_{\mathbf{x}}$, modulo contextual equivalence (including renaming of output and garbage collection), for which we need the following lemma.

*Lemma 7.4* $(\nu a)\,(!\llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} b \mid !\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} a) \;\sim_{\mathrm{C}}\; !\,(\nu a)\,(\llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} b \mid !\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} a)$

*Proof:* By coinduction.

$$
\begin{array}{rcl}
(\nu a)\,(!\llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} b \mid !\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} a) & \equiv & (\nu a)\,(\llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} b \mid !\llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} b \mid !\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} a \mid !\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} a) \\
 & \sim_{\mathrm{C}} & (\nu a)\,(\llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} b \mid !\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} a) \mid (\nu a)\,(!\llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} b \mid !\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} a) \\
(IH) & \sim_{\mathrm{C}} & (\nu a)\,(\llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} b \mid !\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} a) \mid !\,(\nu a)\,(\llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} b \mid !\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} a) \\
 & \equiv & !\,(\nu a)\,(\llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} b \mid !\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} a)
\end{array}
$$
$\square$

Renaming for the full translation is, as above, defined as a subrelation of contextual equivalence; it is justified via the following lemma, which states that we can safely rename the output of an encoded $\lambda$-term. First, we show an auxiliary result; notice that this is, contrary to Lemma 5.5, formulated without reference to the translation, and only concerns synchronisation cells.

*Proposition 7.5* $(\nu xb)\,(c(v,d).(!\,b \twoheadrightarrow v \mid d \twoheadrightarrow e)) \sim_{\mathrm{C}} (\nu a)\,(a \twoheadrightarrow e \mid (\nu xb)\,(c(v,d).(!\,b \twoheadrightarrow v \mid d \twoheadrightarrow a)))$.

We use this result to show the following:

*Lemma 7.6* (Full renaming lemma) *i)* $(\nu a)\,(a \twoheadrightarrow e \mid \llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} a) \;\sim_{\mathrm{C}}\; \llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} e$ .
*ii)* $(\nu a)\,(!\llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} a \mid !\,a \twoheadrightarrow b) \;\sim_{\mathrm{C}}\; !\llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} b$ .

*Proof:* *i)* By induction on the structure of $\lambda$-terms; we only show one case.

$$
\begin{array}{lrcl}
(M = PQ): & (\nu a)\,(a \twoheadrightarrow e \mid \llbracket PQ \rrbracket^{\mathsf{F}}_{\rrbracket} a) & \triangleq & \\
 & (\nu a)\,(a \twoheadrightarrow e \mid (\nu cb)\,(\llbracket P \rrbracket^{\mathsf{F}}_{\rrbracket} c \mid c(v,d).\,(!\,b \twoheadrightarrow v \mid d \twoheadrightarrow a) \mid !\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} b)) & \sim_{\mathrm{C}} & (7.5) \\
 & (\nu cb)\,(\llbracket P \rrbracket^{\mathsf{F}}_{\rrbracket} c \mid c(v,d).\,(!\,b \twoheadrightarrow v \mid d \twoheadrightarrow e) \mid !\llbracket Q \rrbracket^{\mathsf{F}}_{\rrbracket} b) & \triangleq & \llbracket PQ \rrbracket^{\mathsf{F}}_{\rrbracket} e
\end{array}
$$

*ii)* By coinduction and the fact that $!P \triangleq !_\omega P$:

$$
\begin{array}{rcl}
(\nu a)\,(!\llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} a \mid !\,a \twoheadrightarrow b) & \sim_{\mathrm{C}} & (\nu a)\,(\llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} a \mid a \twoheadrightarrow b) \mid (\nu a)\,(!\llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} a \mid !\,a \twoheadrightarrow b) \\
((i)) & \sim_{\mathrm{C}} & \llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} b \mid (\nu a)\,(!\llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} a \mid !\,a \twoheadrightarrow b) \\
(IH) & \sim_{\mathrm{C}} & \llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} b \mid !\llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} b \qquad \equiv\; !\llbracket M \rrbracket^{\mathsf{F}}_{\rrbracket} b
\end{array}
$$
$\square$

$$
\begin{array}{ll}
\llbracket(\lambda x.(\lambda z.(\lambda y.M)x))N\rrbracket^{\mathbf{F}}_{\mathbf{J}}a & \stackrel{\Delta}{=},\equiv \\
(\nu cb)\,((\nu xb_1)\,(\llbracket\lambda z.(\lambda y.M)x\rrbracket^{\mathbf{F}}_{\mathbf{J}}b_1 \mid \overline{c}\langle x,b_1\rangle) \mid c(v,d).\,(!\,b\rightarrow v \mid d\rightarrow a) \mid !\llbracket N\rrbracket^{\mathbf{F}}_{\mathbf{J}}b) & \rightarrow_{\pi}(c) \\
(\nu bxb_1)\,(\llbracket(\lambda z.(\lambda y.M)x)\rrbracket^{\mathbf{F}}_{\mathbf{J}}b_1 \mid !\,b\rightarrow x \mid b_1\rightarrow a \mid !\llbracket N\rrbracket^{\mathbf{F}}_{\mathbf{J}}b) & \sim_{\mathrm{C}} \\
(\nu x)\,(\llbracket\lambda z.(\lambda y.M)x\rrbracket^{\mathbf{F}}_{\mathbf{J}}a \mid !\llbracket N\rrbracket^{\mathbf{F}}_{\mathbf{J}}x) & \stackrel{\Delta}{=},\equiv \\
(\nu x)\,((\nu zb)\,((\nu cb_1)\,(\llbracket\lambda y.M\rrbracket^{\mathbf{F}}_{\mathbf{J}}c \mid c(v,d).\,(!\,b_1\rightarrow v \mid d\rightarrow b) \mid !\llbracket x\rrbracket^{\mathbf{F}}_{\mathbf{J}}b_1) \mid \overline{a}\langle z,b\rangle) \mid !\llbracket N\rrbracket^{\mathbf{F}}_{\mathbf{J}}x) & \stackrel{\Delta}{=} \\
(\nu x)\,((\nu zb)\,((\nu cb_1)\,((\nu yb_2)\,(\llbracket M\rrbracket^{\mathbf{F}}_{\mathbf{J}}b_2 \mid \overline{c}\langle y,b_2\rangle) \mid c(v,d).\,(!\,b_1\rightarrow v \mid d\rightarrow b) \mid !\llbracket x\rrbracket^{\mathbf{F}}_{\mathbf{J}}b_1) \mid \overline{a}\langle z,b\rangle) \mid !\llbracket N\rrbracket^{\mathbf{F}}_{\mathbf{J}}x) & \rightarrow_{\pi}(c_1) \\
(\nu x)\,((\nu zb)\,((\nu b_1)\,((\nu yb_2)\,(\llbracket M\rrbracket^{\mathbf{F}}_{\mathbf{J}}b_2 \mid !\,b_1\rightarrow y \mid b_2\rightarrow b) \mid !\llbracket x\rrbracket^{\mathbf{F}}_{\mathbf{J}}b_1) \mid \overline{a}\langle z,b\rangle) \mid !\llbracket N\rrbracket^{\mathbf{F}}_{\mathbf{J}}x) & \sim_{\mathrm{C}} \\
(\nu x)\,((\nu zb)\,((\nu y)\,(\llbracket M\rrbracket^{\mathbf{F}}_{\mathbf{J}}b \mid !\llbracket x\rrbracket^{\mathbf{F}}_{\mathbf{J}}y) \mid \overline{a}\langle z,b\rangle) \mid !\llbracket N\rrbracket^{\mathbf{F}}_{\mathbf{J}}x) & \equiv \\
(\nu zb)\,((\nu y)\,((\nu x)\,(\llbracket M\rrbracket^{\mathbf{F}}_{\mathbf{J}}b \mid !\llbracket N\rrbracket^{\mathbf{F}}_{\mathbf{J}}x) \mid (\nu x)\,(!\llbracket x\rrbracket^{\mathbf{F}}_{\mathbf{J}}y \mid !\llbracket N\rrbracket^{\mathbf{F}}_{\mathbf{J}}x)) \mid \overline{a}\langle z,b\rangle) & \sim_{\mathrm{C}} \\
(\nu zb)\,((\nu y)\,((\nu x)\,(\llbracket M\rrbracket^{\mathbf{F}}_{\mathbf{J}}b \mid !\llbracket N\rrbracket^{\mathbf{F}}_{\mathbf{J}}x) \mid !\llbracket N\rrbracket^{\mathbf{F}}_{\mathbf{J}}y) \mid \overline{a}\langle z,b\rangle) & \stackrel{\Delta}{=} \\
\llbracket\lambda z.M\,\langle x:=N\rangle\,\langle y:=N\rangle\rrbracket^{\mathbf{F}}_{\mathbf{J}}a &
\end{array}
$$

Figure 6: Running $\llbracket(\lambda x.(\lambda z.(\lambda y.M)x))N\rrbracket^{\mathbf{F}}_{\mathbf{J}}a$.

Since $\llbracket M\langle x:=N\rangle\rrbracket^{\mathbf{H}}_{\mathbf{J}}a$ places $\llbracket M\rrbracket^{\mathbf{H}}_{\mathbf{J}}a$ and $\llbracket N\rrbracket^{\mathbf{H}}_{\mathbf{J}}x$ in parallel, we can even show that the $\lambda\mathbf{x}$-variant of the Substitution Lemma $P[Q/y][R/x] = P[R/x][Q[R/x]/y]$ is preserved:

*Lemma 7.7* (Substitution Lemma) $\llbracket P\langle y:=Q\rangle\langle x:=R\rangle\rrbracket^{\mathbf{F}}_{\mathbf{J}}a \;\sim_{\mathrm{C}}\; \llbracket P\langle x:=R\rangle\langle y:=Q\langle x:=R\rangle\rangle\rrbracket^{\mathbf{F}}_{\mathbf{J}}a$ .

*Proof:*
$$
\begin{array}{rrl}
\llbracket P\langle y:=Q\rangle\langle x:=R\rangle\rrbracket^{\mathbf{F}}_{\mathbf{J}}a & \stackrel{\Delta}{=} & (\nu x)\,((\nu y)\,(\llbracket P\rrbracket^{\mathbf{F}}_{\mathbf{J}}a \mid !\llbracket Q\rrbracket^{\mathbf{F}}_{\mathbf{J}}y) \mid !\llbracket R\rrbracket^{\mathbf{F}}_{\mathbf{J}}x) \\
 & \sim_{\mathrm{C}} & (\nu y)\,((\nu x)\,(\llbracket P\rrbracket^{\mathbf{F}}_{\mathbf{J}}a \mid !\llbracket R\rrbracket^{\mathbf{F}}_{\mathbf{J}}x) \mid (\nu x)\,(!\llbracket Q\rrbracket^{\mathbf{F}}_{\mathbf{J}}y \mid !\llbracket R\rrbracket^{\mathbf{F}}_{\mathbf{J}}x)) \\
 & (7.4)\sim_{\mathrm{C}} & (\nu y)\,((\nu x)\,(\llbracket P\rrbracket^{\mathbf{F}}_{\mathbf{J}}a \mid !\llbracket R\rrbracket^{\mathbf{F}}_{\mathbf{J}}x) \mid !\,(\nu x)\,(\llbracket Q\rrbracket^{\mathbf{F}}_{\mathbf{J}}y \mid !\llbracket R\rrbracket^{\mathbf{F}}_{\mathbf{J}}x)) \\
 & \stackrel{\Delta}{=} & (\nu y)\,(\llbracket P\,\langle x:=R\rangle\rrbracket^{\mathbf{F}}_{\mathbf{J}}a \mid !\llbracket Q\,\langle x:=R\rangle\rrbracket^{\mathbf{F}}_{\mathbf{J}}y) \\
 & \stackrel{\Delta}{=} & \llbracket P\langle x:=R\rangle\langle y:=Q\langle x:=R\rangle\rangle\rrbracket^{\mathbf{F}}_{\mathbf{J}}a
\end{array}
$$

We could consider again $\rightsquigarrow^{*}_{\pi} = \rightarrow^{*}_{\pi},\sim_{\mathrm{G}},\sim_{\mathrm{R}}$ , taking into account that $\sim_{\mathrm{R}}$ would now be defined differently, but that gives a problem in the case of an application in the proof of Theorem 7.8, since we cannot reduce under replication. However, $\rightsquigarrow^{*}_{\pi} \subseteq \sim_{\mathrm{C}}$, and by Proposition 2.5 the latter is respected by replication, so this is the right relation for our result.

As in Theorem 5.9, we can now show a reduction preservation result for full explicit reduction for $\lambda\mathbf{x}$, by showing that $\llbracket\cdot\rrbracket^{\mathbf{F}}_{\mathbf{J}}\cdot$ preserves $\rightarrow_{\mathbf{x}}$ up to $\sim_{\mathrm{C}}$. This effectively shows that our interpretation respects reduction on terms, and shows that that has no effect on the observable behaviour.

**Theorem 7.8** $M \rightarrow_{\mathbf{x}} N \;\Rightarrow\; \llbracket M\rrbracket^{\mathbf{F}}_{\mathbf{J}}a \sim_{\mathrm{C}} \llbracket N\rrbracket^{\mathbf{F}}_{\mathbf{J}}a$.

*Proof:* By induction on explicit reduction; we focus on the base cases.

$$
\begin{array}{rrl}
((\lambda x.M)P \rightarrow M\,\langle x:=P\rangle\,):\; \llbracket(\lambda x.M)P\rrbracket^{\mathbf{F}}_{\mathbf{J}}a & \stackrel{\Delta}{=} & \\
(\nu cb)\,((\nu xb_1)\,(\llbracket M\rrbracket^{\mathbf{F}}_{\mathbf{J}}b_1 \mid \overline{c}\langle x,b_1\rangle) \mid c(v,d).\,(!\,b\rightarrow v \mid d\rightarrow a) \mid !\llbracket P\rrbracket^{\mathbf{F}}_{\mathbf{J}}b) & \rightarrow_{\pi}(c) & \\
(\nu cb)\,(\llbracket M\rrbracket^{\mathbf{F}}_{\mathbf{J}}b_1 \mid !\,b\rightarrow x \mid b_1\rightarrow a \mid !\llbracket P\rrbracket^{\mathbf{F}}_{\mathbf{J}}b) & \sim_{\mathrm{R}} & \\
(\nu x)\,(\llbracket M\rrbracket^{\mathbf{F}}_{\mathbf{J}}a \mid !\llbracket P\rrbracket^{\mathbf{F}}_{\mathbf{J}}x) & \stackrel{\Delta}{=} & \llbracket M\langle x:=P\rangle\rrbracket^{\mathbf{F}}_{\mathbf{J}}a \\
((MN)\,\langle x:=P\rangle \rightarrow M\,\langle x:=P\rangle N\,\langle x:=P\rangle\,):\; \llbracket MN\langle x:=P\rangle\rrbracket^{\mathbf{F}}_{\mathbf{J}}a & \stackrel{\Delta}{=} & \\
(\nu x)\,((\nu cb)\,(\llbracket M\rrbracket^{\mathbf{F}}_{\mathbf{J}}c \mid c(v,d).\,(!\,b\rightarrow v \mid d\rightarrow a) \mid !\llbracket N\rrbracket^{\mathbf{F}}_{\mathbf{J}}b) \mid !\llbracket P\rrbracket^{\mathbf{F}}_{\mathbf{J}}x) & \sim_{\mathrm{C}} & \\
(\nu cb)\,((\nu x)\,(\llbracket M\rrbracket^{\mathbf{F}}_{\mathbf{J}}c \mid !\llbracket P\rrbracket^{\mathbf{F}}_{\mathbf{J}}x) \mid c(v,d).\,(!\,b\rightarrow v \mid d\rightarrow a) \mid (\nu x)\,(!\llbracket N\rrbracket^{\mathbf{F}}_{\mathbf{J}}b \mid !\llbracket P\rrbracket^{\mathbf{F}}_{\mathbf{J}}x)) & \sim_{\mathrm{C}} & (7.4) \\
(\nu cb)\,((\nu x)\,(\llbracket M\rrbracket^{\mathbf{F}}_{\mathbf{J}}c \mid !\llbracket P\rrbracket^{\mathbf{F}}_{\mathbf{J}}x) \mid c(v,d).\,(!\,b\rightarrow v \mid d\rightarrow a) \mid !\,(\nu x)\,(\llbracket N\rrbracket^{\mathbf{F}}_{\mathbf{J}}b \mid !\llbracket P\rrbracket^{\mathbf{F}}_{\mathbf{J}}x)) & \stackrel{\Delta}{=} & \\
\llbracket M\langle x:=P\rangle N\langle x:=P\rangle\rrbracket^{\mathbf{F}}_{\mathbf{J}}a & &
\end{array}
$$

$$
\begin{array}{ll}
\llbracket x((\lambda z.z)(\lambda y.y))\rrbracket^{\mathsf{F}}_{\mathsf{J}} a & \stackrel{\Delta}{=} \\
(\nu cb)\,(\llbracket x\rrbracket^{\mathsf{F}}_{\mathsf{J}} c \mid c(v,d).\,(!b \rightarrowtriangle v \mid d \rightarrowtriangle a) \mid !\llbracket(\lambda z.z)(\lambda y.y)\rrbracket^{\mathsf{F}}_{\mathsf{J}} b) & \equiv \\
(\nu cb)\,(\llbracket x\rrbracket^{\mathsf{F}}_{\mathsf{J}} c \mid c(v,d).\,(!b \rightarrowtriangle v \mid d \rightarrowtriangle a) \mid \llbracket(\lambda z.z)(\lambda y.y)\rrbracket^{\mathsf{F}}_{\mathsf{J}} b \mid !\llbracket(\lambda z.z)(\lambda y.y)\rrbracket^{\mathsf{F}}_{\mathsf{J}} b) & \stackrel{\Delta}{=} \\
(\nu cb)\,(\llbracket x\rrbracket^{\mathsf{F}}_{\mathsf{J}} c \mid c(v,d).\,(!b \rightarrowtriangle v \mid d \rightarrowtriangle a) \mid (\nu c_1 b_1)\,((\nu z b_2)\,(\llbracket z\rrbracket^{\mathsf{F}}_{\mathsf{J}} b_2 \mid \overline{c_1}\langle z,b_2\rangle) \mid & \\
\qquad c_1(v,d).\,(!b_1 \rightarrowtriangle v \mid d \rightarrowtriangle b) \mid !\llbracket\lambda y.y\rrbracket^{\mathsf{F}}_{\mathsf{J}} b_1) \mid !\llbracket(\lambda z.z)(\lambda y.y)\rrbracket^{\mathsf{F}}_{\mathsf{J}} b) & \rightarrow_\pi (c_1) \\
(\nu cb)\,(\llbracket x\rrbracket^{\mathsf{F}}_{\mathsf{J}} c \mid c(v,d).\,(!b \rightarrowtriangle v \mid d \rightarrowtriangle a) \mid (\nu b_1)\,((\nu z b_2)\,(\llbracket z\rrbracket^{\mathsf{F}}_{\mathsf{J}} b_2 \mid !b_1 \rightarrowtriangle z \mid b_2 \rightarrowtriangle b) \mid & \\
\qquad !\llbracket\lambda y.y\rrbracket^{\mathsf{F}}_{\mathsf{J}} b_1) \mid !\llbracket(\lambda z.z)(\lambda y.y)\rrbracket^{\mathsf{F}}_{\mathsf{J}} b) & \equiv, \stackrel{\Delta}{=} \\
(\nu cb)\,(\llbracket x\rrbracket^{\mathsf{F}}_{\mathsf{J}} c \mid c(v,d).\,(!b \rightarrowtriangle v \mid d \rightarrowtriangle a) \mid (\nu b_1)\,((\nu z b_2)\,(z(w).\,\overline{b_2}\langle w\rangle \mid !b_1 \rightarrowtriangle z \mid b_2 \rightarrowtriangle b) \mid & \\
\qquad (\nu y b_3)\,(\llbracket y\rrbracket^{\mathsf{F}}_{\mathsf{J}} b_3 \mid \overline{b_1}\langle y,b_3\rangle) \mid !\llbracket\lambda y.y\rrbracket^{\mathsf{F}}_{\mathsf{J}} b_1) \mid !\llbracket(\lambda z.z)(\lambda y.y)\rrbracket^{\mathsf{F}}_{\mathsf{J}} b) & \rightarrow_\pi (b_1,z,b_2) \\
(\nu cb)\,(\llbracket x\rrbracket^{\mathsf{F}}_{\mathsf{J}} c \mid c(v,d).\,(!b \rightarrowtriangle v \mid d \rightarrowtriangle a) \mid (\nu y b_3)\,(\llbracket y\rrbracket^{\mathsf{F}}_{\mathsf{J}} b_3 \mid \overline{b}\langle y,b_3\rangle) \mid & \\
\qquad (\nu b_1 z)\,(!b_1 \rightarrowtriangle z \mid !\llbracket\lambda y.y\rrbracket^{\mathsf{F}}_{\mathsf{J}} b_1) \mid !\llbracket(\lambda z.z)(\lambda y.y)\rrbracket^{\mathsf{F}}_{\mathsf{J}} b) & \stackrel{\Delta}{=}, \sim_{\mathrm{C}} \\
(\nu cb)\,(\llbracket x\rrbracket^{\mathsf{F}}_{\mathsf{J}} c \mid c(v,d).\,(!b \rightarrowtriangle v \mid d \rightarrowtriangle a) \mid \llbracket\lambda y.y\rrbracket^{\mathsf{F}}_{\mathsf{J}} b \mid !\llbracket(\lambda z.z)(\lambda y.y)\rrbracket^{\mathsf{F}}_{\mathsf{J}} b) &
\end{array}
$$

Figure 7: Simulating contraction of a redex in the right-hand side of an application.

$$
\begin{array}{lll}
((\lambda y.M)\,\langle x{:=}P\rangle \rightarrow \lambda y.(M\,\langle x{:=}P\rangle)):\ \llbracket(\lambda y.M)\,\langle x{:=}P\rangle\rrbracket^{\mathsf{F}}_{\mathsf{J}} a & \stackrel{\Delta}{=} & \\
\quad (\nu x)\,((\nu y b)\,(\llbracket M\rrbracket^{\mathsf{F}}_{\mathsf{J}} b \mid \overline{a}\langle y,b\rangle) \mid !\llbracket P\rrbracket^{\mathsf{F}}_{\mathsf{J}} x) & \equiv & \\
\quad (\nu y b)\,((\nu x)\,(\llbracket M\rrbracket^{\mathsf{F}}_{\mathsf{J}} b \mid !\llbracket P\rrbracket^{\mathsf{F}}_{\mathsf{J}} x) \mid \overline{a}\langle y,b\rangle) & \stackrel{\Delta}{=} & \llbracket\lambda y.M\,\langle x{:=}P\rangle\rrbracket^{\mathsf{F}}_{\mathsf{J}} a \\
(x\,\langle x{:=}P\rangle \rightarrow P):\ \llbracket x\,\langle x{:=}P\rangle\rrbracket^{\mathsf{F}}_{\mathsf{J}} a & \stackrel{\Delta}{=} & \\
\quad (\nu x)\,(\llbracket x\rrbracket^{\mathsf{F}}_{\mathsf{J}} a \mid !\llbracket P\rrbracket^{\mathsf{F}}_{\mathsf{J}} x) & \equiv & \\
\quad (\nu x)\,(x(w).\,\overline{a}\langle w\rangle \mid \llbracket P\rrbracket^{\mathsf{F}}_{\mathsf{J}} x \mid !\llbracket P\rrbracket^{\mathsf{F}}_{\mathsf{J}} x) & \sim_{\mathrm{C}} (7.6) & \\
\quad (\nu x)\,(\llbracket P\rrbracket^{\mathsf{F}}_{\mathsf{J}} a \mid !\llbracket P\rrbracket^{\mathsf{F}}_{\mathsf{J}} x) & \equiv & \\
\quad \llbracket P\rrbracket^{\mathsf{F}}_{\mathsf{J}} a \mid (\nu x)\,(!\llbracket P\rrbracket^{\mathsf{F}}_{\mathsf{J}} x) & \sim_{\mathrm{G}} & \llbracket P\rrbracket^{\mathsf{F}}_{\mathsf{J}} a \\
(M\,\langle x{:=}P\rangle \rightarrow M,\ x \notin fv(M)):\ \llbracket M\langle x{:=}P\rangle\rrbracket^{\mathsf{F}}_{\mathsf{J}} a & \stackrel{\Delta}{=} & \\
\quad (\nu x)\,(\llbracket M\rrbracket^{\mathsf{F}}_{\mathsf{J}} a \mid !\llbracket P\rrbracket^{\mathsf{F}}_{\mathsf{J}} x) & \equiv & (x \notin fn(\llbracket M\rrbracket^{\mathsf{F}}_{\mathsf{J}} a)) \\
\quad \llbracket M\rrbracket^{\mathsf{F}}_{\mathsf{J}} a \mid (\nu x)\,(!\llbracket P\rrbracket^{\mathsf{F}}_{\mathsf{J}} x) & \sim_{\mathrm{G}} & \llbracket M\rrbracket^{\mathsf{F}}_{\mathsf{J}} a
\end{array}
$$

The contextual rules follow by induction, using Proposition 2.5. □

Since $(\nu a)\,(\overline{a}\langle b\rangle \mid a(x).\,P) \rightarrow_\pi P[b/x]$ does not imply $!\,(\nu a)\,(\overline{a}\langle b\rangle \mid a(x).\,P) \rightarrow_\pi !\,P[b/x]$, but only implies $!\,(\nu a)\,(\overline{a}\langle b\rangle \mid a(x).\,P) \sim_{\mathrm{C}} !\,P[b/x]$, we are forced to formulate our result with $\sim_{\mathrm{C}}$. Moreover, in the proof of this last result, in the case for application we needed to call on Lemma 7.4.

We now give an illustration to this result:

*Example 7.9* Take the term $(\lambda x.(\lambda z.(\lambda y.M)x))N$; we can reduce this term as follows:

$$
\begin{array}{rl}
(\lambda x.(\lambda z.(\lambda y.M)x))N & \rightarrow_{\mathbf{x}}\ (\lambda z.(\lambda y.M)x)\,\langle x{:=}N\rangle \\
& \rightarrow_{\mathbf{x}}\ \lambda z.((\lambda y.M)x)\,\langle x{:=}N\rangle \\
& \rightarrow_{\mathbf{x}}\ \lambda z.(\lambda y.M)\,\langle x{:=}N\rangle(x\,\langle x{:=}N\rangle) \\
& \rightarrow_{\mathbf{x}}\ \lambda z.(\lambda y.M\,\langle x{:=}N\rangle)(x\,\langle x{:=}N\rangle) \\
& \rightarrow_{\mathbf{x}}\ \lambda z.(\lambda y.M\,\langle x{:=}N\rangle)N \\
& \rightarrow_{\mathbf{x}}\ \lambda x.M\,\langle x{:=}N\rangle\,\langle y{:=}N\rangle
\end{array}
$$

We can emulate this reduction using the full encoding, as shown in Figure 6, which clearly shows that we can emulate more than explicit head-reduction.

Since $\lambda\mathbf{x}$ implements $\beta$-reduction, faithful to the principle we introduced, we can show the main criterion for our interpretation:

**Theorem 7.10** (Operational Soundness) *i) If $M \rightarrow^*_\beta N$, then $\llbracket M\rrbracket^{\mathsf{F}}_{\mathsf{J}} a \sim_{\mathrm{C}} \llbracket N\rrbracket^{\mathsf{F}}_{\mathsf{J}} a$.*

*ii ) If $M\uparrow$, then $\llbracket M \rrbracket^{\mathrm{F}}_{\mathrm{J}} a\uparrow$.*

*Proof:* The first is shown by induction using Theorem 7.8; the second follows from Example 7.2. $\square$

Since contextual equivalence is symmetric, this result states that our translation gives, in fact, a semantics for the $\lambda$-calculus:

*Corollary 7.11* (Adequacy) *If $M =_\beta N$, then $\llbracket M \rrbracket^{\mathrm{F}}_{\mathrm{J}} a \sim_{\mathrm{C}} \llbracket N \rrbracket^{\mathrm{F}}_{\mathrm{J}} a$.*

We cannot show "if $\llbracket M \rrbracket^{\mathrm{F}}_{\mathrm{J}} a \sim_{\mathrm{C}} \llbracket N \rrbracket^{\mathrm{F}}_{\mathrm{J}} a$, then $M =_\beta N$" (the reverse of soundness), since different unsolvable terms like $(\lambda x.xx)(\lambda x.xx)$ and $(\lambda y.yyy)(\lambda y.yyy)$ are contextually equivalent, but not $\beta$-equivalent. But since their interpretations under $\llbracket \cdot \rrbracket^{\mathrm{F}}_{\mathrm{J}} \cdot$ never exhibit an output, they are contextually equivalent. This is standard for models of $\beta$-reduction,

Moreover, our notion of type assignment catches this: since $\llbracket N \rrbracket^{\mathrm{F}}_{\mathrm{J}} a$ has at most only one visible output (which is $a$), the process $(\nu a)\,(!\llbracket N \rrbracket^{\mathrm{F}}_{\mathrm{J}} a)$ has *no* visible output, and can therefore be typed by $(\nu a)\,(!\llbracket N \rrbracket^{\mathrm{F}}_{\mathrm{J}} a) : \Gamma \vdash_\pi \emptyset$, *i.e.* with an empty right-hand context; this means that the type system is capable of indicating those processes that can be ignored and stopped from running, which could be used for a type-based reduction. We leave this for future research.

To conclude this paper, and to illustrate the expressiveness of our translation, we give some examples:

*Example 7.12* First, we can run in the right-hand side of an application, as shown in Figure 7. Notice that we had to spawn a copy of the replicated right-hand side first. So we indeed model more than just lazy or spine reduction. We can extend that by showing by induction:

$$\begin{array}{l}(\nu cb)\,(\llbracket x \rrbracket^{\mathrm{F}}_{\mathrm{J}} c \mid c(v,d).\,(!\,b \rightarrowtail v \mid d \rightarrowtail a) \mid \llbracket \lambda y.y \rrbracket^{\mathrm{F}}_{\mathrm{J}} b \mid !\llbracket (\lambda z.z)(\lambda y.y) \rrbracket^{\mathrm{F}}_{\mathrm{J}} b) \;\sim_{\mathrm{C}} \\ (\nu cb)\,(\llbracket x \rrbracket^{\mathrm{F}}_{\mathrm{J}} c \mid c(v,d).\,(!\,b \rightarrowtail v \mid d \rightarrowtail a) \mid !\llbracket \lambda y.y \rrbracket^{\mathrm{F}}_{\mathrm{J}} b)\end{array}$$

But we can do more:

$$\begin{array}{ll}\llbracket (\lambda x.M)((\lambda z.z)(\lambda y.N)) \rrbracket^{\mathrm{F}}_{\mathrm{J}} a & \triangleq \\ (\nu cb)\,((\nu x b_1)\,(\llbracket M \rrbracket^{\mathrm{F}}_{\mathrm{J}} b_1 \mid \overline{c}\langle x,b_1\rangle) \mid c(v,d).\,(!\,b \rightarrowtail v \mid d \rightarrowtail a) \mid !\llbracket (\lambda z.z)(\lambda y.N) \rrbracket^{\mathrm{F}}_{\mathrm{J}} b) & \rightarrow_\pi \;\; (\textit{as above}) \\ (\nu b x b_1)\,(\llbracket M \rrbracket^{\mathrm{F}}_{\mathrm{J}} b_1 \mid \llbracket \lambda y.N \rrbracket^{\mathrm{F}}_{\mathrm{J}} x \mid b_1 \rightarrowtail a \mid !\llbracket (\lambda z.z)(\lambda y.N) \rrbracket^{\mathrm{F}}_{\mathrm{J}} b \mid !\,b \rightarrowtail x) & \triangleq \\ (\nu b x b_1)\,(\llbracket M \rrbracket^{\mathrm{F}}_{\mathrm{J}} b_1 \mid (\nu y b')\,(\llbracket N \rrbracket^{\mathrm{F}}_{\mathrm{J}} b' \mid \overline{x}\langle y,b'\rangle) \mid b_1 \rightarrowtail a \mid !\llbracket (\lambda z.z)(\lambda y.N) \rrbracket^{\mathrm{F}}_{\mathrm{J}} b \mid !\,b \rightarrowtail x) & \end{array}$$

as shown by this reduction, which is similar to the first, we can reduce $\llbracket (\lambda x.M)((\lambda z.z)(\lambda y.N)) \rrbracket^{\mathrm{F}}_{\mathrm{J}} a$ to a process that contains $\llbracket M \rrbracket^{\mathrm{F}}_{\mathrm{J}} b_1 \mid (\nu y b')\,(\llbracket N \rrbracket^{\mathrm{F}}_{\mathrm{J}} b' \mid \overline{x}\langle y,b'\rangle)$, *i.e.* where in one spawned-off copy of the process $!\llbracket (\lambda z.z)(\lambda y.N) \rrbracket^{\mathrm{F}}_{\mathrm{J}} b$ we have 'contracted the redex $(\lambda z.z)(\lambda y.N)$'. The variables $x$ in $M$ are all encoded via $x(w).\overline{e_i}\langle w\rangle$, for some $\vec{e_i}$, and do not occur under any guard, so any communication over $x$ can take place, generating a process that is like $\llbracket M \rrbracket^{\mathrm{F}}_{\mathrm{J}} b_1$ but for the fact that one $x(w).\overline{e_i}\langle w\rangle$ is replaced by $\llbracket \lambda y.N \rrbracket^{\mathrm{F}}_{\mathrm{J}} e_i$. Of course we can repeat this procedure of 'spawning and contraction' as often as needed, and show that $\llbracket (\lambda x.M)((\lambda z.z)(\lambda w.N)) \rrbracket^{\mathrm{F}}_{\mathrm{J}} a \rightarrow_\pi \llbracket M[\lambda y.N/x] \rrbracket^{\mathrm{F}}_{\mathrm{J}} b_1 \mid b_1 \rightarrowtail a$ so can model the reduction $(\lambda x.M)((\lambda z.z)(\lambda y.N)) \rightarrow_\beta M[\lambda y.N/x]$ via a reduction that is not lazy, nor spine-reduction.

## Conclusions and Future Work

We have found new, simple and intuitive translations of $\lambda$-terms in $\pi$ that respects head reduction with explicit substitution, and $b$-equality, respectively. For this translation, we have shown that termination is preserved. We have shown that, for our context assignment system that uses the type constructor $\rightarrow$ for $\pi$ and is based on classical logic, typeable $\lambda$-terms are interpreted by our translation as typeable $\pi$-processes, preserving the types.

## References


[1] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit Substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[2] 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.

[3] 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.

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

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

[6] S. van Bakel and M. Fernández. Normalisation, Approximation, and Semantics for Combinator Systems. *Theoretical Computer Science*, 290:975–1019, 2003.

[7] 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 Italian Conference on Theoretical Computer Science* (ICTCS'05), *Siena, Italy*, volume 3701 of *Lecture Notes in Computer Science*, pages 81–96. Springer Verlag, 2005.

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

[9] S. van Bakel and M.G. Vigliotti. A logical interpretation of the $\lambda$-calculus into the $\pi$-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.

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

[11] 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.

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

[13] 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.

[14] A. Church. A Note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1(1):40–41, 1936.

[15] M. Cimini, C. Sacerdoti Coen, and D. Sangiorgi. $\overline{\lambda}\mu\tilde{\mu}$ calculus, $\pi$-calculus, and abstract machines. In *Proceedings of EXPRESS'09*, 2009.

[16] H.B. Curry. Grundlagen der Kombinatorischen Logik. *American Journal of Mathematics*, 52:509–536, 789–834, 1930.

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

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

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

[20] 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.

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

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

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

[24] S.B. Lassen. Head Normal Form Bisimulation for Pairs and the $\lambda\mu$-Calculus. In *Proceedings of 21th IEEE Symposium on Logic in Computer Science (LICS'06), 12-15 August 2006, Seattle, WA, USA*, pages 297–306, 2006.

[25] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):269–310, 1992.
[26] R. Milner. *Communicating and Mobile Systems: the $\pi$-calculus*. Cambridge University Press, 1999.
[27] 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.
[28] B.C. Pierce and D. Sangiorgi. Typing and Subtyping for Mobile Processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996.
[29] D. Sangiorgi. *Expressing Mobility in Process Algebra: First Order and Higher Order Paradigms*. PhD thesis, Edinburgh University, 1992.
[30] 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.
[31] D. Sangiorgi. Lazy functions and mobile processes. Rapport de Recherche 2515, INRIA, Sophia-Antipolis, France, 1995.
[32] D. Sangiorgi and D. Walker. *The Pi-Calculus*. Cambridge University Press, 2001.
[33] P. Sestoft. Standard ML on the Web server. Department of Mathematics and Physics, Royal Veterinary and Agricultural University, Denmark, 1996.
[34] P. Sestoft. Demonstrating Lambda Calculus Reduction. In *The Essence of Computation*, volume 2566 of *Lecture Notes in Computer Science*, pages 420–435. Springer-Verlag, 2002.
[35] H. Thielecke. *Categorical Structure of Continuation Passing Style*. PhD thesis, University of Edinburgh, 1997. LFCS technical report ECS-LFCS-97-376.
[36] C. Urban. *Classical Logic and Computation*. PhD thesis, University of Cambridge, October 2000.
[37] C. Urban and G.M. Bierman. Strong normalisation of cut-elimination in classical logic. *Fundamenta Informaticae*, 45(1,2):123–155, 2001.
[38] 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.