λ to π: encodings, duality

Jean-Marie Madiot
Training period at LIP, ENS Lyon, équipe Plume

July 31, 2011

Le contexte général

Le π-calcul est un modèle pour les systèmes concurrents et un exemple de calcul de processus, sa particularité étant d'être par *passage de nom*. C'est un modèle très expressif, et en particulier Turing-complet: plusieurs encodages du λ-calcul vers le π-calcul existent, tout d'abord par Milner dès l'introduction du π-calcul [Mil92], puis des approches unifiées selon les stratégies d'évaluation, par Sangiorgi et Walker [SW01]. D'un autre côté, le π-calcul jouit d'une dualité entrée/sortie dans un certain fragment décrit par Sangiorgi [San96], Boreale [Bor98] ou encore Merro [MS04].

Le problème étudié

Généralement les encodages λ vers π existants étaient dans une certaine mesure plus orientés *entrée* que *sortie*, c'est à dire que les fonctions encodées sont reliées à un canal de retour qui est utilisé en tant que canal réceptif. Mais plus récemment, certains encodages [vBV09], [vBV10] se révélèrent d'orientation inverse. La question de la dualité entrée/sortie entre ces encodages se pose alors naturellement, mais malheureusement ils ne font pas partie du fragment dualisant du π-calcul, donc cette remarque n'a même pas de formulation précise.

Une telle dualité permettrait de mettre en évidence les caractéristiques communes à de tels encodages, donc dans une certaine mesure étudier les caractéristiques nécessaires à un fragment du π-calcul pour obtenir l'expressivité calculatoire du λ-calcul.

La contribution proposée

Les encodages de λ vers π ont une image qui n'est pas dans le fragment symétrique de π qu'on peut dualiser, qu'on appelle πI. Des traductions d'un fragment de π vers une extension de πI existent, mais ce fragment est trop petit pour les encodages.

Dans un premier temps on ajoute alors à πI un cas particulier de définition récursive, des *liens* qui ont la capacité de gérer à la fois des canaux réceptifs et des canaux émetteurs. Ainsi, on montre que cette extension de πI a une expressivité suffisante pour considérer les encodages de λ vers π, qui sont alors dans une certaine mesure duaux l'un de l'autre, par l'intermédiaire de la traduction de π vers πI.

Les propriétés de la traduction ont mis à rude épreuve les définitions du typage, des liens et de la traduction elle-même, ainsi que les notions mêmes d'équivalence mises en jeu dans le résultat. Finalement chaque contre-exemple raffinait le fragment candidat pour la traduction dans πI, si bien qu'on a ajouté au π-calcul des contraintes de typage puis d'asynchronie partielle.

Dans un deuxième temps, comme la dualité dans l'extension de πI existante n'est pas totalement satisfaisante, on introduit une extension πw de π qui contient un nouvel opérateur, le *fil*. Après l'étude de ce calcul on peut alors traduire πw dans son fragment dualisant πwI, dans lequel la dualité des encodages de λ vers π s'exprime plus naturellement.

Les arguments en faveur de sa validité

L'expressivité de l'extension de πI considérée premièrement se manifeste par une traduction de π vers πI, qui jouit en fait d'une propriété de *full abstraction*. Ce résultat fait que la dualité dans πI induit une notion de dualité dans π, qui est celle dont on avait besoin initialement. Cette *full abstraction* traite d'équivalences manipulant d'une façon particulière les noms, à la manière de [MS04], mais il reste encore à éclaircir un point technique dans sa preuve.

L'extension πw considérée deuxièmement est en revanche plus satisfaisante, car la traduction de πw dans πwI n'introduit pas les liens qui sont des processus techniquement encombrants impliqués dans πI. On a étudié ici la théorie de πIw qui permet alors de s'approcher plus vite d'une réponse définitive à la question, mais elle doit encore devenir un peu plus mature pour établir une dualité qui a du sens.

Le bilan et les perspectives

Il reste bien sûr à régler le point technique de la traduction de π vers πI.

Il reste également à développer la théorie de πw, ainsi que celle de πwI, même indépendamment des questions de dualité qui nous intéressaient ici. On aimerait notamment comparer les expressivités de πwI et de π.

Enfin des travaux connexes suggèrent d'étudier les liens avec le calcul des fusions de [PV98], les *equators* de [Mer99], les types de [BHY05], l'utilisation des *linear forwarders* dans [GLW03] et les autres encodages de [Vas05]. On développe ces liens en section 6.1.

1 Introduction

The π-calculus is a model for concurrent systems. It is a process calculus that can *pass names*. This name-passing process calculus have the particularity to be able to express the λ-calculus and thus is Turing-complete. However some aspects of name-passing may not be needed for this expressiveness and we would like to determine which they are. Moreover there are several ways to encode the λ-calculus into the π-calculus. The usual encodings consider functions as receiving processes, but [vBV10] introduced an encoding based on functions as outputs. The latter appears similar to one of the formers, apart from the orientation of the channels which seems switched. We may want to express the fact that there are dual to each other, but the present tools for duality do not allow us to formulate such a statement.

We will broaden the expressiveness of the dualizing fragment of the π-calculus by translating a bigger fragment of the π-calculus into it in order to express the notion of duality.

1.1 Process calculi

The π-calculus is a name-passing process calculus and a common model for concurrent systems. Its processes are terms described by the following grammar, reminded as a convention for notations. The terminal a range over a countable set of *names* and $\tilde{x}$, $\tilde{b}$ are tuples of names.

$$P ::= 0 \mid P|P \mid a(\tilde{x}).P \mid \overline{a}\langle\tilde{b}\rangle.P \mid (\nu a)P \mid !a(\tilde{x}).P$$

We will also sometimes use the following term constructors:

$$P ::= \cdots \mid a(\tilde{x}) : P \mid \overline{a}(\tilde{b}).P \mid \overline{a}(\tilde{b}) : P$$

On these terms we consider the usual operational semantics defined by the usual late labelled transition system: (where α means any replicable prefix, i.e. $a(x)$ or $\overline{a}(x)$ for future calculi that will allow it)

$$\frac{P \xrightarrow{\overline{a}b} P' \quad Q \xrightarrow{a(x)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'[b/x]}\ comm_l \qquad \frac{P \xrightarrow{\overline{a}(b)} P' \quad Q \xrightarrow{a(x)} Q'}{P \mid Q \xrightarrow{\tau} (\nu b)(P' \mid Q'[b/x])}\ close_l$$

$$\frac{P \xrightarrow{\mu} P' \quad \mathrm{bn}(\mu) \cap \mathrm{fn}(R) = \emptyset}{P \mid R \xrightarrow{\mu} P' \mid R}\ par_l \qquad \frac{P \xrightarrow{\mu} P' \quad x \notin \mathrm{n}(\mu)}{(\nu x)P \xrightarrow{\mu} (\nu x)P'}\ new$$

$$\frac{P \xrightarrow{\overline{a}x} P' \quad a \neq x}{(\nu x)P \xrightarrow{\overline{a}(x)} P'}\ open \qquad \frac{}{\alpha.P \xrightarrow{\alpha} P}\ pre \qquad \frac{}{!\alpha.P \xrightarrow{\alpha} P \mid !\alpha.P}\ bang$$

We will also also use the delayed input, noted $a(\tilde{x}) : P$ which has less restrictions than the usual input prefix, and also authorizes both actions through the prefix and communications between the prefix and the suffix:

$$\frac{}{\alpha : P \xrightarrow{\alpha} P}\ Din \qquad \frac{P \xrightarrow{\mu} P' \quad \mathrm{bn}(\alpha) \notin \mathrm{n}(\mu)}{\alpha : P \xrightarrow{\mu} \alpha : P}\ Dact$$

$$\frac{P \xrightarrow{\overline{a}(b)} P'}{a(x) : P \xrightarrow{\tau} (\nu b)P[b/x]}\ Dcomm \qquad \frac{P \xrightarrow{a(b)} P'}{\overline{a}(x) : P \xrightarrow{\tau} (\nu b)P[b/x]}\ Dclose$$

We will consider several fragments of the π-calculus. We keep a list of the calculi we use in appendix A.1.

- $A\pi$, *the asynchronous π-calculus*, is the subset of the π-calculus where outputs are always followed by the null process: $\overline{a}b.0$ ($\overline{a}b$ for short).
- $L\pi$, *the localized π-calculus*, the subset of π with the restriction that a process receiving a name cannot use it in input subject position
- $AL\pi = A\pi \cap L\pi$

1.2 λ-calculus and evaluation strategies

The λ-calculus is a model of computation based on term substitution, we recall its grammar, where x ranges over a countable set of *variables*, and the β-reduction, where $M[n/x]$ means M where all occurrences of x are replaced by N:

$$M ::= \lambda x.M \mid MM \mid x$$

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

The β-reduction can take place anywhere in the term and this forms a rewriting system. If we restrict this to one or less locations, we get an evaluation strategy. In the following we will use the call-by-name strategy:

$$\frac{M \to_\beta M'}{MN \to_\beta M'N} \qquad \frac{}{(\lambda x.M)N \to_\beta M[N/x]}$$

And we will also use the *strong* call-by-name, or *spine*, i.e. call-by-name in addition to the following rule. In this system β-reductions can take place under λ-abstractions. Note that it is not an evaluation strategy, as the term $(\lambda z.(\lambda x.x)z)y$ has more than one derivative.

$$\frac{M \to_\beta M'}{\lambda x.M \to_\beta \lambda x.M'}$$

1.3 Encodings of λ into π

The first encoding of the λ-calculus into the π-calculus was conceived by Robin Milner [Mil92]. Given a λ-term M and a name p, it builds a process that can be called on the channel p representing the function M. We will call p the *return channel*. The arguments of this call are x, the argument of the function and q, the return channel for the result of the encoding.

$$[\![\lambda x.M]\!]^{\mathcal{M}}_p = p(x,q).[\![M]\!]^{\mathcal{M}}_q$$

The body of the function can ask for its argument x to be evaluated to the channel p:

$$[\![x]\!]^{\mathcal{M}}_p = \overline{x}p$$

The function application uses these definitions to call the function M on q, asking to put the result on p, and giving the fresh name x for the argument. Then each time the body of the function M asks for its argument on some channel r, the replicated "argument server" will answer to it:

$$[\![MN]\!]^{\mathcal{M}}_p = (\nu q)([\![M]\!]^{\mathcal{M}}_q \mid (\nu x)(\overline{q}\langle x, p\rangle.!x(r).[\![N]\!]^{\mathcal{M}}_r))$$

These three equations define an encoding of the λ-calculus with the call-by-name evaluation strategy [Mil92]. There exist also slightly more verbose encodings of call-by-name λ-calculus described in [SW01] that are easier to study, since they are in Lπ:

$$\begin{aligned}
[\![\lambda x.M]\!]^{\mathcal{N}}_p &= \overline{p}(y).y(x, q).[\![M]\!]^{\mathcal{N}}_q \\
[\![x]\!]^{\mathcal{N}}_p &= \overline{x}p \\
[\![MN]\!]^{\mathcal{N}}_p &= (\nu q)([\![M]\!]^{\mathcal{N}}_q \mid q(y).(\nu x)(\overline{y}\langle x, p\rangle.!x(r).[\![N]\!]^{\mathcal{N}}_r))
\end{aligned}$$

The only difference is that the return name is used to send a fresh name y representing a pointer to the function that waits for the argument channel and the return channel.

In those two encodings a prefix input prevents the evaluation under the λ-abstraction. Replacing this prefix input by a delayed input will have the direct consequence on the translated evaluation strategy: call-by-name becomes *strong* call-by-name. We note $[\![\cdot]\!]^{\mathcal{M}_S}$ Milner's encoding with this small modification on the abstraction:

$$[\![\lambda x.M]\!]^{\mathcal{M}_S}_p = p(x, q) : [\![M]\!]^{\mathcal{M}_S}_q$$

and we note $[\![\cdot]\!]^{\mathcal{N}_S}$ similarly:

$$[\![\lambda x.M]\!]^{\mathcal{N}_S}_p = \overline{p}(y) : y(x, q) : [\![M]\!]^{\mathcal{N}_S}_q$$

Output-based encodings All the previous encodings, as well as the uniform encodings seen in [SW01] (which interpret other evaluation strategies), represent functions as *receiving* processes: they begin by an input. [vBV10] builds an new encoding noted here $[\![\cdot]\!]^{\mathcal{O}}_p$ where functions are represented by *emitting* processes, so this encoding is new and not input-based like the others:

$$\begin{aligned}
[\![x]\!]^{\mathcal{O}}_p &= x(p').p' \twoheadrightarrow p \\
[\![\lambda x.M]\!]^{\mathcal{O}}_p &= \overline{p}(x, q) : [\![M]\!]^{\mathcal{O}}_q \\
[\![MN]\!]^{\mathcal{O}}_p &= (\nu q)([\![M]\!]^{\mathcal{O}}_q \mid q(x, p').(p' \twoheadrightarrow p \mid !\overline{x}(r).[\![N]\!]^{\mathcal{O}}_r))
\end{aligned}$$

Where $a \twoheadrightarrow b$, called a *finite link*, is short for $a(x).\overline{b}x$.

However we remark here that this encoding is close to Milner's encoding, the main difference is that inputs and outputs seem to have been inverted. There are other differences, though: one cannot invert inputs and outputs in a free output. We observe that the process corresponding to the free output $\overline{x}p$ is $x(p').p' \twoheadrightarrow p$, an input followed by a link. This link connects the name supposed to be the object of the output to the received name. We can observe this mechanism in the encodings of the variable and of the application.

This encoding is introduced in [vBV10] with a type system and Curry types. One of its purposes is to relate the π-calculus to Curry types and both intuitionistic and classical logic. We ignore this type system and these aspects in this report and we focus on the encoding itself.

2 i/o types and internal mobility

In this section, we present tools and notions we will use in the following of this report. Expansion relations (section 2.1) refine the notion of bisimulation and allow proof techniques on weak bisimulations. i/o types (section 2.2) allow a tighter control on the source language. The fragment πI (section 2.3) will be used as the dualizing fragment of π.

2.1 Bisimulations and expansions

We call the processes P' such that $P \xrightarrow{\mu} P'$ for some label μ the derivatives of P. We define the following transition relations:

- $\Longrightarrow$ is $\xrightarrow{\tau}{}^{*}$, the reflexive transitive closure of $\xrightarrow{\tau}$
- $\overset{\mu}{\Longrightarrow}$ is $\Longrightarrow\xrightarrow{\mu}\Longrightarrow$
- $\overset{\hat{\mu}}{\Longrightarrow}$ is $\Longrightarrow$ (if $\mu = \tau$) or $\overset{\mu}{\Longrightarrow}$ (otherwise)
- $\xrightarrow{\hat{\mu}}$ is $\xrightarrow{\tau} \cup =$ (if $\mu = \tau$) or $\xrightarrow{\mu}$ (otherwise)
- $\xleftarrow{\mu}$ (resp. $\overset{\mu}{\Longleftarrow}$, ...) is the inverse relation of $\xrightarrow{\mu}$ (resp. $\overset{\mu}{\Longrightarrow}$, ...)
- $\mathcal{R}\mathcal{S}$ is the composition of $\mathcal{R}$ and $\mathcal{S}$ i.e. $\{(a,c)\ \exists b, a\mathcal{R}b \wedge b\mathcal{S}c\}$

Definition 2.1. A *strong bisimulation* on a set of processes $\mathcal{P}$ is a relation $\mathcal{R}$ on $\mathcal{P}$ such that, for all action μ, $(\xleftarrow{\mu} \mathcal{R}) \subseteq (\mathcal{R} \xleftarrow{\mu})$ (i.e. whenever $P\mathcal{R}Q$ and $P \xrightarrow{\mu} P'$, there is a Q' such that $P'\mathcal{R}Q'$ and $Q \xrightarrow{\mu} Q'$) and $(\mathcal{R} \xrightarrow{\mu}) \subseteq (\xrightarrow{\mu} \mathcal{R})$ (i.e. whenever $P\mathcal{R}Q$ and $Q \xrightarrow{\mu} Q'$, there is a P' such that $P'\mathcal{R}Q'$ and $P \xrightarrow{\mu} P'$). The strong bisimilarity $\sim$ is the union of all strong bisimulations.

Definition 2.2. A *weak bisimulation* on a set of processes $\mathcal{P}$ is a relation $\mathcal{R}$ on $\mathcal{P}$ such that, for all action μ, $(\xleftarrow{\mu} \mathcal{R}) \subseteq (\mathcal{R} \overset{\mu}{\Longleftarrow})$ and $(\mathcal{R} \xrightarrow{\mu}) \subseteq (\overset{\mu}{\Longrightarrow} \mathcal{R})$. The weak bisimilarity $\approx$ is the union of all weak bisimulations.

Definition 2.3. A *expansion* on a set of processes $\mathcal{P}$ is a relation $\mathcal{R}$ on $\mathcal{P}$ such that, for all action μ, $(\xleftarrow{\mu} \mathcal{R}) \subseteq (\mathcal{R} \overset{\mu}{\Longleftarrow})$ and $(\mathcal{R} \xrightarrow{\mu}) \subseteq (\xrightarrow{\hat{\mu}} \mathcal{R})$. $\lesssim$ is the union of all expansions.

Informally $P \lesssim Q$ means that Q behaves like P but can do more τ-transitions than P can. It is an asymmetric intermediate between strong and weak bisimilarities: $\sim \subsetneq \lesssim \subsetneq \approx$.

2.2 i/o types

Types are usually used to prevent runtime errors, like arity mismatches in polyadic π-calculus. Types usually provide some roles to variables, terms or processes in order to categorize them. The i/o type systems assign input and output roles for the name in the π-calculus.

The following type system, which we will refer to as π_{io}, is inspired from the standard i/o types described in [PS96]. The type constructor $\#T$ (a subtype of iT and oT) is now allowed only when introduced by a restriction $(\nu a)P$. The type of x in $a(x).P$ will be a *strict* type, i.e. only composed of is and os.

The strict types (T) are defined below. μ is a fixed point constructor and allows "infinite" types (finite types defined by the grammar below quotiented by the smallest congruence relation containing $\mu x.T = T[\mu x.T/x]$). The non-strict types S have also one hybrid constructor $\#$ at top-level, $\#T$ meaning both iT and oT.

$$T ::= i\tilde{T} \mid o\tilde{T} \mid \mu x.T \tag{1}$$

$$S ::= T \mid \#\tilde{T} \tag{2}$$

The typing judgments are of the form $\Gamma \vdash P$ for the processes and $\Gamma \vdash x : S$ for the names, where Γ is a context, represented by a finite map from the names to the non-strict types. There is no implicit subtyping except for the names: the types of the form $\#T$ are used to type names in the context and the inference rules for names can only change the head of the type:

$$\frac{\Gamma(x) \in \{iT, \#T\}}{\Gamma \vdash x : iT} \qquad \frac{\Gamma(x) \in \{oT, \#T\}}{\Gamma \vdash x : oT}$$

The rules to type processes against contexts are the usual rules for the parallel operations (where α is as usual $a(x)$ or $\overline{a}(x)$):

$$\frac{}{\Gamma \vdash 0} \qquad \frac{\Gamma \vdash \alpha.P}{\Gamma \vdash !\alpha.P} \qquad \frac{\Gamma \vdash P_1 \quad \Gamma \vdash P_2}{\Gamma \vdash P_1 \mid P_2}$$

The interesting rules are about name manipulation. They are different from the usual rules about the kind of the types of the names, i.e. whether they are strict or not: the names introduced by the restriction may not be strict, but the variables introduced by an input have to.

$$\frac{\Gamma, a : \#T \vdash P}{\Gamma \vdash (\nu a)P} \qquad \frac{\Gamma \vdash a : oT \quad \Gamma \vdash v : T \quad \Gamma \vdash P}{\Gamma \vdash \overline{a}\langle v \rangle.P}$$

$$\frac{\Gamma \vdash a : iT \quad \Gamma, x : T \vdash P}{\Gamma \vdash a(x).P} \qquad \frac{\Gamma \vdash a : iT \quad \Gamma, x : T \vdash P}{\Gamma \vdash a(x) : P}$$

We will consider π_{io} as a calculus itself, since this type system is preserved by any labelled reduction. Moreover $L\pi$ is a subcalculus of π_{io}:

Lemma 2.1 (Subject reduction)**.** If $\Gamma \vdash P$ and $P \xrightarrow{\mu} P'$ then $\Gamma' \vdash P'$ (with $\mathrm{dom}(\Gamma') = \mathrm{dom}(\Gamma) + \mathrm{bn}(\mu)$)

Proof in appendix A.3. □

Lemma 2.2 ($\mathrm{L}\pi \subset \pi_{io}$)**.** If P is a process of $\mathrm{L}\pi$, then P is typable in π_{io} with the type $\#o^\omega$, where $o^\omega := \mu T.oT$:

$$\mathrm{fn}(P) : \#o^\omega \vdash P$$

Proof in appendix A.3. □

2.3 Internal mobility

πI In the π-calculus the main operational rule $\overline{a}\langle b\rangle.P \mid a(x).Q \xrightarrow{\tau} P \mid Q[b/x]$ involves carrying names – from the emitting process to the receiving process – and aliasing, making it less simple than CCS at an implementation level. An intermediate solution is the π-calculus with internal mobility [San96] which allows only the emission of private names. Internal mobility mainly relies on α-conversion, which is easier to manipulate than global names.

This calculus, called πI, is the full π-calculus with the constrain that every output $\overline{a}b.P$ is bound or private, that is, preceded by (νb). We will use the shortcut $\overline{a}(b).P$ for $(\nu b)\overline{a}b.P$. The grammar of πI is then as follows:

$$P ::= 0 \mid P|P \mid (\nu a)P \mid a(\tilde{x}).P \mid \overline{a}(\tilde{b}).P \mid !a(\tilde{x}).P \mid !\overline{a}(\tilde{x}).P$$

πI, as well as its operational semantics, enjoy a duality between inputs and outputs. Namely, if $\overline{a(x)} = \overline{a}(x)$, $\overline{\overline{a}(x)} = a(x)$, $\overline{\tau} = \tau$ for both prefixes and labelled transitions, then

$$P \xrightarrow{\mu} P' \text{ iff } \overline{P} \xrightarrow{\overline{\mu}} \overline{P'}$$

Links [Bor98] shows that πI can fully express the asynchronous localized π-calculus, ALπ. This result requires the introduction the *links* – which are a special case of recursive definition – to transform a free output prefix into a bound output prefix.

A link is recursively defined as follows (The shortcut $\tilde{y} \hookrightarrow \tilde{x}$ stands for $y_1 \hookrightarrow x_1 \mid y_2 \hookrightarrow x_2 \mid \dots$):

$$a \hookrightarrow b := !a(\tilde{x}).\overline{b}(\tilde{y}).\tilde{y} \hookrightarrow \tilde{x}$$

The cardinality of $\tilde{x}$ and $\tilde{y}$ is determined by the arity of a, with the help of the environment. This is possible if the polyadic π-calculus is sorted. For a calculus that is not sorted, we would need an infinite product of links, one for every possible arity.

In order to simulate a free output of a process not in πI, the role of the link is to connect the name supposed to be sent to the new and private name of the bound output in πI, so $\overline{a}b$ will be transformed into the πI process $\overline{a}(x).x \hookrightarrow b$.

Translation of ALπ into πI These links allow the definition of this translation [Bor98] from ALπ to πI, which is non-trivial only for the (asynchronous) output.

$$[\![0]\!]^I_{AL} = 0 \qquad [\![P \mid Q]\!]^I_{AL} = [\![P]\!]^I_{AL} \mid [\![Q]\!]^I_{AL} \qquad [\![!\alpha.P]\!]^I_{AL} = ![\![\alpha.P]\!]^I_{AL}$$

$$[\![a(\tilde{x}).P]\!]^I_{AL} = a(\tilde{x})[\![P]\!]^I_{AL} \qquad [\![\overline{a}\langle\tilde{b}\rangle]\!]^I_{AL} = \overline{a}(\tilde{x}).\tilde{x} \hookrightarrow \tilde{b} \qquad [\![(\nu a)P]\!]^I_{AL} = (\nu a)[\![P]\!]^I_{AL}$$

This translation is proven to be a full abstraction for some notion of barbed bisimilarity in [Bor98] and for some notion of bisimilarity in [MS04]. For the curious reader, the appendix A.2 explains in details how the links work inside the translation.

Limitations These links allow a translation from ALπ to πI. They allow the encoding of the transmission of the output capability over the channels: in $\overline{a}b \mid a(x).P$ as x cannot be used in input subject position in P. Typing the fragment ALπ amounts to always giving the type o^ω or $\#o^\omega$ to any name. We would like to also translate the transmission of the input capability into πI. This would let us consider processes with richer i/o types, for example some encodings of the λ-calculus into the π-calculus that are not in ALπ.

3 Generalizing the translation from ALπ to πI

From now on we will generalize the existing translation, with ALπ as source language, to obtain a translation with π_{io} as source language. The following contains most of the contributions of the internship.

3.1 Definitions

3.1.1 Dualizing links

Adapting the translation into πI to i/o-typed processes brings some difficulties. For example, one needs to consider $a(x).P$ even when x is used in input subject position in P, so the usual link would not work. Indeed the link cannot communicate with an input since it starts with one. For example, with $P = x(z).0$ (the first line is the expected behavior, the second is the (bad) attempt of translation):

$$\begin{array}{rcll} \overline{a}b \mid a(x).x(z).0 & \xrightarrow{\tau} & b(z).0 & \xrightarrow{b(z)} \quad 0 \\ \overline{a}(y).y \hookrightarrow b \mid a(x).x(z).0 & \xrightarrow{\tau} & (\nu y)(y(u).\overline{b}(v).u \hookrightarrow v \mid y(z).0) & \not\rightarrow \end{array}$$

The immediate conclusion is that the way the passed name is used, that is, as an input or an output, is very significant for building the corresponding link.

We introduce a different type of link that depends on a given type T that has to be strict[1]:

$$a \overset{oT_1}{\hookrightarrow} b := !a(x).\overline{b}(y).y \overset{T_1}{\hookrightarrow} x$$
$$a \overset{iT_1}{\hookrightarrow} b := !\overline{a}(x).b(y).x \overset{T_1}{\hookrightarrow} y$$

Like the links used in the translation from ALπ into πI, these links are always infinite processes. Unlike them, they have a replicated bound output in addition to the replicated input, but that goes well with the duality in πI. Note that the definition of the link that is used in the translation of ALπ into πI corresponds to this definition with the type o^ω: $\hookrightarrow = \overset{o^\omega}{\hookrightarrow}$.

Remark 3.1 (Polyadicity). We can easily adapt the definition to a polyadic calculus by replacing the link in the suffix by a product of links, depending of the type. For example:

$$a \overset{o(T_1,T_2)}{\hookrightarrow} b := !a(x_1,x_2).\overline{b}(y_1,y_2).(y_1 \overset{T_1}{\hookrightarrow} x_1 \mid y_2 \overset{T_2}{\hookrightarrow} x_2)$$

3.1.2 π_{io} to πI

Source and target calculi The translation into πI aims to be as general as possible, but the types of π_{io} are needed for the links. The source language of the translation will be π_{io} with both delayed and usual input prefixes, as well as synchronous outputs. Both input prefixes and bound output prefixes can be replicated.

The target calculus is πI also with both delayed and usual input prefixes, as well as synchronous (bound) outputs. Both bound output and input prefixes can be replicated, and we authorize recursive definitions (or just links).

[1] a strict type contains no $\#T$, see equation (1) for the grammar of strict types

The translation itself The translation is essentially the same as the one from ALπ to πI [Bor98]. The main difference is that this one have to take into consideration the fact that the translated processes are typable in π_{io}, so the translation would have a second argument that is the typing derivation of the considered process.

In the type system π_{io} from a derivation tree for a given process we can always easily recover a derivation tree for any of its subterms. Thanks to this compositionality, we will consider the typing derivation an implicit argument.

$$[\![0]\!]^I_{io} = 0 \qquad [\![P \mid Q]\!]^I_{io} = [\![P]\!]^I_{io} \mid [\![Q]\!]^I_{io} \qquad [\![!\alpha.P]\!]^I_{io} = ![\![\alpha.P]\!]^I_{io}$$

$$[\![a(\tilde{x}).P]\!]^I_{io} = a(\tilde{x})[\![P]\!]^I_{io} \quad [\![\overline{a}\langle\tilde{b}\rangle.P]\!]^I_{io} = \overline{a}(\tilde{x}).(\tilde{x} \overset{\tilde{T}}{\hookrightarrow} \tilde{b} \mid [\![P]\!]^I_{io}) \quad [\![(\nu a)P]\!]^I_{io} = (\nu a)[\![P]\!]^I_{io}$$

The only difference with $[\![-]\!]^I_{AL}$ is in the translation of the free output: the output can be synchronous, and the link needs a list of strict types. These types are the remaining of the type $T(a)$ of a in the typing derivation without its top-level connective. Namely: $\tilde{T}$ if $T(a) = \#\tilde{T}$ or $T(a) = o\tilde{T}$. The derivation also provide the fact that the arity of $\tilde{T}$ is the same as the arity of $\tilde{b}$.

As said before, the links $\hookrightarrow$ that are used in $[\![-]\!]^I_{AL}$ are a special case of our links: $\hookrightarrow = \overset{o^\omega}{\hookrightarrow}$. Therefore on L$\pi$ the two translations coincide, since L$\pi \subset \pi_{io}$.

An optimization However, the translation of $(\nu b)\overline{a}b.P$ can be $\overline{a}(b).[\![P]\!]^I_{io}$ instead of the naive translation $(\nu b)\overline{a}(b').(b' \overset{T}{\hookrightarrow} b \mid [\![P]\!]^I_{io})$, since the output is already bound. We will also consider an optimization for such cases:

$$[\![(\nu b)\overline{a}b.P]\!'^I_{io} = \overline{a}(b).[\![P]\!'^I_{io}$$

And $[\![P]\!'^I_{io}$ is defined like $[\![P]\!]^I_{io}$ each time P does not match this pattern. The advantage of this optimization, apart from the lightness, is that the link will not induce asynchronous behaviors in the translated suffix $[\![P]\!'^I_{io}$.

3.2 Properties of the translation

In order to prove a full abstraction theorem we will need several laws about this translation. These laws are also used as sanity checks before investigating the final result. The transitivity law is used each time a link interacts with another one in the translated processes. It is also is used in the lemmas afterwards. The substitution lemma is the easiest way to develop a correct version of the links, as it is simpler and more demanding than a result of full abstraction.

Lemma 3.1 (transitivity law)**.** If $b \neq a, c$ then

$$\forall T \; a \overset{T}{\hookrightarrow} c \lesssim (\nu b)(a \overset{T}{\hookrightarrow} b \mid b \overset{T}{\hookrightarrow} c)$$

Proof in appendix A.3. We manually close the relation by the intended transitions. □

We also need a replication law allowing the distributivity of a replicated process over parallel composition, in order to distribute links over processes of the form $P \mid Q$.

Lemma 3.2 (replication law)**.** If

1. a is the subject of the prefix α
2. no derivative of P_1 can communicate with a derivative of P_2 about[2] a
3. no derivative of R can make an action on a

then

$$(\nu a)(\alpha.R \mid P_1 \mid P_2) \sim (\nu a)(\alpha.R \mid P_1) \mid (\nu a)(\alpha.R \mid P_2)$$

Proof in appendix A.3. The corresponding relation is a strong bisimulation up to ν contexts. □

Corollary 3.1. If b is fresh, T is a strict type and $\Gamma, a : T \vdash P_1, P_2$ then

$$(\nu a)(a \overset{T}{\hookrightarrow} b \mid [\![P_1]\!] \mid [\![P_2]\!]) \sim (\nu a)(a \overset{T}{\hookrightarrow} b \mid [\![P_1]\!]) \mid (\nu a)(a \overset{T}{\hookrightarrow} b \mid [\![P_2]\!])$$

Proof. The conditions of lemma 3.2 are checked. 1. and 3. are straightforward. For 2. either $T = o\tilde{T_1}$ or $T = i\tilde{T_1}$. Either way P_1 and P_2 will only be able to receive or emit on a and not both, so they will not communicate on a. And thanks to the fact that for all Q, $[\![Q]\!]$ cannot send the (free) name a, this also work for all their derivatives. □

Then we can begin to tackle the substitution law which states that the parallel composition of a link and a process behaves like the latter in which we replaced the source of the link by its target.

We first define the notion of a substitutable name in a given process.

Definition 3.1. The *substitutability* of a in P ($P \ / \ a$) is the smallest predicate satisfying the following rules.

$$\frac{}{0 \ / \ a} \quad \frac{P \ / \ a \quad Q \ / \ a}{P \mid Q \ / \ a} \quad \frac{\alpha.P \ / \ a}{!\alpha.P \ / \ a}$$

$$\frac{P \ / \ a \quad P \ / \ x}{a(x) : P \ / \ a} \quad \frac{}{\overline{a}b \ / \ a} \quad \frac{c \neq a \quad P \ / \ a}{\overline{c}b.P \ / \ a} \quad \frac{c \neq a \quad P \ / \ a}{c(x).P \ / \ a} \quad \frac{c \neq a \quad P \ / \ a}{c(x) : P \ / \ a}$$

In short a is substitutable in P if it does not appear in synchronous subject position or in non-delayed input subject position. But in the case of $a(x) : P$, substituting a with a link will lead to substitute x in P with another link, so we request the substitutability of x in P.

Several results in the following are based on the following lemma, but its proof is not complete. For clarity we add a star (*) on results with incomplete proofs and all the results depending on them.

Lemma 3.3 (substitution law*)**.** If a is substitutable in P, let T be any type for which there exists Γ such that $\Gamma, a : T \vdash P$. Then:

$$[\![P]\!]^I_{io}[b/a] \lesssim (\nu a)(a \overset{T}{\hookrightarrow} b \mid [\![P]\!]^I_{io})$$

Proof in appendix A.3. Proof sketch: By induction on the size of P, we prove that:
if $(\Gamma, \tilde{a} : \tilde{T} \vdash P)$ and ($\tilde{a}$ are substitutable in P) then

$$[\![P]\!][\tilde{b}/\tilde{a}] \lesssim (\nu \tilde{a})(\tilde{a} \overset{\tilde{T}}{\hookrightarrow} \tilde{b} \mid [\![P]\!])$$

$$b(x) : [\![P]\!] \lesssim (\nu x)(b(y).x \overset{T}{\hookrightarrow} y \mid [\![P]\!]) \text{ if } x \in \tilde{a}$$

This proof is not complete, see details in appendix. □

[2] i.e. with actions neither of subject a nor of object a

3.3 Full abstraction

The usual method to prove that a translation is correct is to prove a full abstraction result. The translation $T : \mathcal{P}_1 \to \mathcal{P}_2$ is fully abstract for the equivalence relations $=_1$ and $=_2$ if, for all P and Q, we have the following equivalence: $(P =_1 Q)$ iff $(T(P) =_2 T(Q))$.

The full abstraction is not directly possible for usual bisimilarities, since the translation introduces links that never die. These links make it hard to relate two processes when one of them could use a link to connect two names.

The following definition handles correctly the notion of bisimilarity in presence of links. Indeed when an output occurs a link remain in the process. This bisimilarity is introduced in [MS04] and we modify it to cope with the optimization of the translation: in $[\![\overline{a}(b).P]\!] = \overline{a}(b).[\![P]\!]$ then the transition $\overline{a}(b)$ does not necesseraly imply a link.

Definition 3.2 (Semi link bisimilarity)**.** A symmetric relation $\mathcal{R}$ is a semi-link bisimulation on processes typable in π_{io} if, whenever $P \mathcal{R} Q$,

1. If $P \xrightarrow{\tau} P'$ then $Q \Longrightarrow Q'$ and $P' \mathcal{R} Q'$
2. If $P \xrightarrow{a(x)} P'$ $(x \notin \mathrm{fn}(Q))$ then
   (a) either $Q \overset{a(x)}{\Longrightarrow} Q'$ and $P' \mathcal{R} Q'$
   (b) or $Q \Longrightarrow Q'$ and $P' \mathcal{R} (Q' \mid \overline{a}x)$
3. If $P \xrightarrow{\overline{a}b} P'$, let T be tail$(T(a))$ and p be fresh, then:
   (a) either $Q \overset{\overline{a}d}{\Longrightarrow} Q'$ and $(p \overset{T}{\hookrightarrow} b \mid P') \mathcal{R} (p \overset{T}{\hookrightarrow} d \mid Q')$
   (b) either $Q \overset{\overline{a}(c)}{\Longrightarrow} Q'$ and $(p \overset{T}{\hookrightarrow} b \mid P') \mathcal{R} (\nu c)(p \overset{T}{\hookrightarrow} c \mid Q')$
   (c) or $Q \overset{\overline{a}b}{\Longrightarrow} Q'$ and $P' \mathcal{R} Q'$
4. If $P \xrightarrow{\overline{a}(c)} P'$, let T be tail$(T(a))$ and p be fresh, then:
   (a) either $Q \overset{\overline{a}b}{\Longrightarrow} Q'$ and $(\nu c)(p \overset{T}{\hookrightarrow} c \mid P') \mathcal{R} (p \overset{T}{\hookrightarrow} b \mid Q')$
   (b) either $Q \overset{\overline{a}(c)}{\Longrightarrow} Q'$ and $(\nu c)(p \overset{T}{\hookrightarrow} c \mid P') \mathcal{R} (\nu c)(p \overset{T}{\hookrightarrow} c \mid Q')$
   (c) or $Q \overset{\overline{a}(c)}{\Longrightarrow} Q'$ and $P' \mathcal{R} Q'$

Where tail$(T(a))$ is the remaining of the type of a in the context when we remove its top-level connective. P and Q are semi-link bisimilar $(P \approx_l Q)$ if they are related by some semi-link bisimulation.

The following bisimilarity is the ground asynchronous bisimilarity but on translated processes.

Definition 3.3 ($\asymp_a$-bisimilarity)**.** A symmetric relation $\mathcal{R}$ on processes of πI is a $\asymp_a$-bisimulation if whenever $P \mathcal{R} Q$:

1. If $P \xrightarrow{\tau} P'$ then $Q \Longrightarrow Q'$ and $P' \mathcal{R} Q'$

2. If $P \xrightarrow{\overline{a}(b)} P'$ $(b \notin \mathrm{fn}(Q))$ then $Q \overset{\overline{a}(b)}{\Longrightarrow} Q'$ and $P' \mathcal{R} Q'$
3. If $P \xrightarrow{a(b)} P'$ $(b \notin \mathrm{fn}(Q))$ then
   (a) either $Q \overset{\overline{a}(b)}{\Longrightarrow} Q'$ and $P' \mathcal{R} Q'$
   (b) or $Q \Longrightarrow Q'$ and $P' \mathcal{R} (Q' \mid [\![\overline{a}b]\!])$

P and Q are $\asymp_a$-bisimilar if they related by some $\asymp_a$-bisimulation.

Lemma 3.4 (Step lemma*)**.** Let P be in π_{io} and that $|c|$ is the type carried by the channel of name c in the environment. (here, $[\![\cdot]\!]$ represents $[\![\cdot]\!]^I_{io}$)

1. if $P \xrightarrow{a(b)} P'$ then $[\![P]\!] \xrightarrow{a(b)} \gtrsim [\![P']\!]$
2. if $P \xrightarrow{\tau} P'$ then $[\![P]\!] \xrightarrow{\tau} \gtrsim [\![P']\!]$
3. if $P \xrightarrow{\overline{a}b} P'$ then $[\![P]\!] \xrightarrow{\overline{a}(p)} \gtrsim (p \overset{|a|}{\hookrightarrow} b \mid [\![P']\!])$ with $p \# P'$
4. if $P \xrightarrow{\overline{a}(b)} P'$ then
   (a) either $[\![P]\!] \xrightarrow{\overline{a}(p)} \gtrsim (\nu b)(p \overset{|a|}{\hookrightarrow} b \mid [\![P']\!])$ with $p \# P'$
   (b) or $[\![P]\!] \xrightarrow{\overline{a}(b)} \gtrsim [\![P']\!]$

Also:

1. if $[\![P]\!] \xrightarrow{a(x)} P_1$ then $P \xrightarrow{a(x)} P'$ with $P_1 \gtrsim [\![P']\!]$
2. if $[\![P]\!] \xrightarrow{\overline{a}(x)} P_1$ then
   (a) either $P \xrightarrow{\overline{a}b} P'$ with $P_1 \gtrsim (x \overset{|a|}{\hookrightarrow} b \mid [\![P']\!])$ with $x \# P'$
   (b) or $P \xrightarrow{\overline{a}(b)} P'$ with $P_1 \gtrsim (\nu b)(x \overset{|a|}{\hookrightarrow} b \mid [\![P']\!])$ with $x \# P'$
   (c) or $P \xrightarrow{\overline{a}(x)} P'$ with $P_1 \gtrsim [\![P']\!]$
3. if $[\![P]\!] \xrightarrow{\tau} P_1$ then $P \xrightarrow{\tau} P'$ with $P_1 \gtrsim [\![P']\!]$

Proof. The proof profusely relies on the substitution law 3.3 and the transitivity law 3.1, and is proved by induction on the derivation of $P \xrightarrow{\mu} P'$. □

Lemma 3.5 (Full abstraction*)**.** If P and Q are two processes typable in π_{io} then

$$P \approx_l Q \text{ iff } [\![P]\!]'^I_{io} \asymp_a [\![Q]\!]'^I_{io}$$

Proof. We show that $\mathcal{R}_e$ is a $\asymp_a$-bisimulation up to $\gtrsim$ and $\approx$, then we show that $\mathcal{R}_d$ is a link bisimulation.

$$\mathcal{R}_e = \{([\![P]\!]'^I_{io}, [\![Q]\!]'^I_{io}) \mid P \approx_l Q\}$$
$$\mathcal{R}_d = \{(P, Q) \mid [\![P]\!]'^I_{io} \asymp_a [\![Q]\!]'^I_{io}\}$$

Assuming the step lemma 3.4 the proof follows the many cases of the definitions of $\asymp_a$ and $\approx_l$. □

4 From λ to πI

This translation from π_{io} into πI needs some typing information in the source language. Fortunately, the strong call-by-name version of Milner's encoding $[\![\cdot]\!]^{\mathcal{M}_S}$ and the encoding from [vBV10] $[\![\cdot]\!]^{\mathcal{O}}$ are both typable in π_{io} for any λ-term. For convenience we introduce the following tuple type:

$$\gamma := \mu X.(oiX, iX)$$

The translated terms are typable with respectively $oi\gamma$, $ii\gamma$ for the variables of the λ-calculus and $i\gamma$, $o\gamma$ for the return types with respectively the encodings $[\![\cdot]\!]^{\mathcal{M}_S}$, $[\![\cdot]\!]^{\mathcal{O}}$:

Lemma 4.1.

$$\forall M\ \forall p \quad \mathrm{fv}(M) : ii\gamma,\ p : o\gamma \vdash [\![M]\!]^{\mathcal{O}}_p$$
$$\forall M\ \forall p \quad \mathrm{fv}(M) : oi\gamma,\ p : i\gamma \vdash [\![M]\!]^{\mathcal{M}_S}_p$$

These typability results enable translations into πI: both the usual translation $[\![\cdot]\!]^I_{io}$ and its optimized version $[\![\cdot]\!]'^I_{io}$ of these encodings are correctly defined. For the encodings to respect the conditions of the full abstraction we privilege the optimized version.

$$[\![M]\!]^{\mathcal{M}^{\mathcal{I}}_S}_p := [\![[\![M]\!]^{\mathcal{M}_S}_p]\!]'^I_{io}$$
$$[\![M]\!]^{\mathcal{O}^{\mathcal{I}}}_p := [\![[\![M]\!]^{\mathcal{O}}_p]\!]'^I_{io}$$

The translated encoding $[\![\cdot]\!]^{\mathcal{M}^{\mathcal{I}}_S}$ is presented below. The free outputs are replaced with bound outputs followed by links whereas the bound output on q in the case of the application is not translated into a redundant link scheme thanks to the optimized version.

$$\begin{aligned}
[\![x]\!]^{\mathcal{M}^{\mathcal{I}}_S}_p &:= \overline{x}(p').p' \overset{i\gamma}{\hookrightarrow} p \\
[\![\lambda x.M]\!]^{\mathcal{M}^{\mathcal{I}}_S}_p &:= p(x,q) : [\![M]\!]^{\mathcal{M}^{\mathcal{I}}_S}_q \\
[\![MN]\!]^{\mathcal{M}^{\mathcal{I}}_S}_p &:= (\nu q)([\![M]\!]^{\mathcal{M}^{\mathcal{I}}_S}_q \mid (\overline{q}(x,p').(p' \overset{i\gamma}{\hookrightarrow} p \mid\ !x(r).[\![N]\!]^{\mathcal{M}^{\mathcal{I}}_S}_r))
\end{aligned}$$

The translated encoding $[\![\cdot]\!]^{\mathcal{O}^{\mathcal{I}}}$ is slightly more complicated, since the encoding of a finite link $p' \twoheadrightarrow p := p'(a,b).\overline{p}\langle a,b\rangle$ is $p'(a,b).[\![\overline{p}\langle a,b\rangle]\!]^I_{io}$. But this proves to be exactly $p' \overset{o\gamma}{\hookrightarrow} p$, so we simplify the notation below:

$$\begin{aligned}
[\![x]\!]^{\mathcal{O}^{\mathcal{I}}}_p &= x(p').p' \overset{o\gamma}{\hookrightarrow} p \\
[\![\lambda x.M]\!]^{\mathcal{O}^{\mathcal{I}}}_p &= \overline{p}(x,q) : [\![M]\!]^{\mathcal{O}^{\mathcal{I}}}_q \\
[\![MN]\!]^{\mathcal{O}^{\mathcal{I}}}_p &= (\nu q)([\![M]\!]^{\mathcal{O}^{\mathcal{I}}}_q \mid q(x,p').(p' \overset{o\gamma}{\hookrightarrow} p \mid !\overline{x}(r).[\![N]\!]^{\mathcal{O}^{\mathcal{I}}}_r))
\end{aligned}$$

Definition 4.1. The *dual* of a process of πI (with links) is defined as follows:

$$\begin{array}{rclrcl}
\mathrm{dual}(a(x).P) & := & \overline{a}(x).\mathrm{dual}(P) & \mathrm{dual}(0) & := & 0 \\
\mathrm{dual}(\overline{a}(x).P) & := & a(x).\mathrm{dual}(P) & \mathrm{dual}(P \mid Q) & := & \mathrm{dual}(P) \mid \mathrm{dual}(Q) \\
\mathrm{dual}(a(x) : P) & := & \overline{a}(x) : \mathrm{dual}(P) & \mathrm{dual}(!\alpha.P) & := & !\mathrm{dual}(\alpha.P) \\
\mathrm{dual}(\overline{a}(x) : P) & := & a(x) : \mathrm{dual}(P) & \mathrm{dual}((\nu a)P) & := & (\nu a)\mathrm{dual}(P) \\
\mathrm{dual}(a \overset{oT}{\hookrightarrow} b) & := & a \overset{iT}{\hookrightarrow} b & \mathrm{dual}(a \overset{iT}{\hookrightarrow} b) & := & a \overset{oT}{\hookrightarrow} b
\end{array}$$

dual$(\cdot)$ is clearly self-inverse. It inverts inputs and outputs but on links we have a different definition from what we could think. We do not have the expected correspondence $P \xrightarrow{\mu} P'$ iff $\mathrm{dual}(P) \xrightarrow{\overline{\mu}} \mathrm{dual}(P')$, because in the links we have $\xrightarrow{\overline{\mu_2}}\xrightarrow{\overline{\mu_1}}$ corresponding to $\xrightarrow{\mu_1}\xrightarrow{\mu_2}$.

$$a \overset{ioT}{\hookrightarrow} b \xrightarrow{\overline{a}(x)} \xrightarrow{b(y)} \xrightarrow{x(z)} \xrightarrow{\overline{y}(t)} \ldots$$
$$a \overset{ooT}{\hookrightarrow} b \xrightarrow{a(x)} \xrightarrow{\overline{b}(y)} \xrightarrow{y(t)} \xrightarrow{\overline{x}(z)} \ldots$$

However this definition of duality preserves the $\asymp_a$-bisimilarity:

Lemma 4.2 (dual() is $\asymp_a$-preserving*)**.** For all P in π_{io},

$$[\![P]\!]'^{I}_{io} \asymp_a [\![Q]\!]'^{I}_{io} \quad \text{iff} \quad \mathrm{dual}([\![P]\!]'^{I}_{io}) \asymp_a \mathrm{dual}([\![Q]\!]'^{I}_{io})$$

Proof. Work in progress. The dual processes of the links generate execution traces that are permutations of the dual of the exectution traces of the links, so the correspondence is not straightforward. □

The translation of the encodings are clearly dual of each other, in a syntactical manner:

Fact 4.1 ($[\![\cdot]\!]_p^{\mathcal{M}_S^{\mathcal{I}}}$ is dual to $[\![\cdot]\!]_p^{\mathcal{O}^{\mathcal{I}}}$)**.** For all λ-term M, $\mathrm{dual}([\![M]\!]_p^{\mathcal{M}_S^{\mathcal{I}}}) = [\![M]\!]_p^{\mathcal{O}^{\mathcal{I}}}$

A direct consequence of lemma 4.2 and the previous fact makes the two encodings induce the same equivalence relation among λ-terms, proving that:

Corollary 4.1 (Same equivalence*)**.** For all λ-terms M and N,

$$[\![M]\!]_p^{\mathcal{M}_S} \approx_l [\![N]\!]_p^{\mathcal{M}_S} \iff [\![M]\!]_p^{\mathcal{O}} \approx_l [\![N]\!]_p^{\mathcal{O}}$$

Remark 4.1. All lemmas and corollaries marked with a star (3.4, 3.5, 4.1) depends either on the substitution law (3.3) or the soundness of the duality (4.2). Both lack a complete proof. In the following, we bypass this problem by building a calculus where the duality is easier.

5 Wires

The links in πI are a specific construction to connect two existing names in a calculus where one cannot send a free name. They are infinite recursive processes and quite heavy to handle. In order to study πI and its expressiveness we will consider a calculus with a new operator that connects names.

The main idea is to add an operator $\{\cdot = \cdot\}$, called a *wire* connecting two names, that adds an equation label into the LTS, that can be persistent or not:

$$\{a = b\} \xrightarrow{a=b} \{a = b\} \quad \text{or} \quad \{a = b\} \xrightarrow{a=b} 0$$

and that renames (only) the subject of an action:

$$\frac{P \xrightarrow{\overline{a}c} P' \quad Q \xrightarrow{a=b} Q'}{P \mid Q \xrightarrow{\overline{b}c} P' \mid Q'}$$

This wire operator could replace by simple processes the links which are infinite processes. That could help us understanding the mechanisms behind the encodings from λ, but mainly this allows us to build a calculus in which the translation into the fragment with internal mobility will be much easier to study. With this more practical translation, the duality of the fragment will be more workable.

The study of this calculus even for simple problems like $\equiv \subset \sim$ forces to introduce other labelled transitions. For example consider the associativity in this example : $\{a = b\} \mid a(x) \mid \overline{b}c$. In the first case, a is wired to b and then communicates with $\overline{b}c$. However in the other case $a(x)$ should be able to communicate with $\overline{b}c$ so we need to add a new kind of transition: an incomplete τ-transition, noted $\tau/a = b$, waiting for a wire:

$$\frac{P \xrightarrow{\overline{a}c} P' \quad Q \xrightarrow{b(x)} Q'}{P \mid Q \xrightarrow{\tau/a=b} P' \mid Q'} \qquad \frac{P \xrightarrow{\tau/a=b} P' \quad Q \xrightarrow{a=b} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

The example $(a(x) \mid \{b = c\}) \mid (\{a = b\} \mid \overline{c}d)$ shows that we also need to keep track of the wires even beside the usual labels:

$$\frac{P \xrightarrow{\mu} P' \quad Q \xrightarrow{a=b} Q'}{P \mid Q \xrightarrow{\mu, a=b} P' \mid Q'}$$

We also need multiple equations in labels, thanks to this example $((a(x) \mid \{c = d\}) \mid (\{a = b\} \mid \overline{d}e)) \mid \{b = c\}$. Note that this is important both to $(\tau/a = b)$-transitions and usual $a(x)$- and $\overline{a}b$-transitions.

Also remark that with persistent wires the predicate $P \xrightarrow{a=b} P'$ implies that $P' = P$ so we could introduce a simpler predicate in the LTS that says "P can raise a wire $a = b$" but that would make bisimulations definitions more complicated, so we stick with transitions that also allow ephemeral wires.

5.1 πw, a π-calculus with wires

The calculus is now endowed with the wire operator:

$$P ::= \cdots \mid \{a = b\}$$

We call this calculus πw. The labels of the LTS now feature connections between names: E is called a connection (it is an equivalence relation on name), μ is called a connected label, β is called

a simple label.

$$\mu ::= \beta, E \mid (E)$$
$$\beta ::= a(x) \mid \overline{a}b \mid \overline{a}(x) \mid \tau \mid \tau/a = b$$
$$E ::= (a_1 = \cdots = a_n), \ldots, (z_1 = \cdots = z_m)$$

Then we present the inference rules for the labelled transition system of πw. All the rules for the communication in the parallel composition of two processes have been merged into one to be easier to understand and to study. This rule, *comp*, depends on the composition of two labels μ_1 and μ_2 into a label and a two-hole 'context' that composes the processes and possibly applies substitution[3]: $\mu_1 \circ \mu_2 = (\mu, C[\cdot,\cdot])$, where C is of the form $C = (\nu a)(\cdot_\sigma \mid \cdot_{\sigma'})$. (For convenience we note $\ell_{(\mu,C)} = \mu$ and $\mathcal{C}_{(\mu,C)} = C$) Note that The rule *comp* only applies when $\mu_1 \circ \mu_2$ is defined.

$$\frac{}{\{a=b\} \xrightarrow{(a=b)} \{a=b\}}\ wire \qquad \frac{P \xrightarrow{\mu_1} P' \quad Q \xrightarrow{\mu_2} Q'}{P \mid Q \xrightarrow{\ell_{\mu_1\circ\mu_2}} \mathcal{C}_{\mu_1\circ\mu_2}[P', Q']}\ comp$$

$$\frac{P \xrightarrow{\mu} P' \quad \mathrm{bn}(\mu) \cap \mathrm{fn}(R) = \emptyset}{P \mid R \xrightarrow{\mu} P' \mid R}\ par_l \qquad \frac{P \xrightarrow{\beta,E} P' \quad x \notin \mathrm{n}(\beta)}{(\nu x)P \xrightarrow{\beta,E\setminus\{x\}} (\nu x)P'}\ new$$

$$\frac{P \xrightarrow{\overline{a}x,E} P' \quad a \neq x}{(\nu x)P \xrightarrow{\overline{a}(x),E\setminus\{x\}} P'}\ open \qquad \frac{}{\alpha.P \xrightarrow{\alpha,()} P}\ pre \qquad \frac{}{!\alpha.P \xrightarrow{\alpha,()} P \mid !\alpha.P}\ bang$$

We can add rules using the connections to rename names, but really in the label (β, E) we should consider the set of names quotiented by the equivalence relation E.

$$\frac{P \xrightarrow{\overline{a}c,E} P' \quad aEb}{P \xrightarrow{\overline{b}c,()} P'}\ subst_{out} \qquad \frac{P \xrightarrow{\overline{a}(x),E} P' \quad aEb}{P \xrightarrow{\overline{b}(x),()} P'}\ subst_{bout} \qquad \frac{P \xrightarrow{a(x),E} P' \quad aEb}{P \xrightarrow{b(x),()} P'}\ subst_{in}$$

With the following definitions on connections:

- aEb if a and b belong to the same class in E.
- $(E + F)$ is the unification of the classes of E and F: $E + F := (E \cup F)^*$
- $(\beta \bullet E)$ is defined as follows:
  1. if $\beta = (\tau/a = b)$ and aEb, then $\tau, ()$
  2. (β, E) otherwise

And we define the composition of connected labels $\mu_1 \circ \mu_2$, which is just the composition on simple labels $\beta_1 \mathbin{\overline{\circ}} \beta_2$ 'connected by' the sum on connection $(E + F)$.

$\circ$	(E_2)	β_2, E_2
(E_1)	$(E_1 + E_2)$	$\beta_2 \overline{\bullet} (E_1 + E_2)$
β_1, E_1	$\beta_1 \overline{\bullet} (E_1 + E_2)$	$(\beta_1 \mathbin{\overline{\circ}} \beta_2) \overline{\bullet} (E_1 + E_2)$

[3] so it is more than a usual context – which cannot substitute names

The application $\overline{\bullet}$ just applies the connection E to the label and not the context:

$$(\mu, C) \mathbin{\overline{\bullet}} E = (\mu \bullet E, C)$$

And the composition $\overline{\circ}$ on simple labels corresponds to the usual composition of labels, but here we give the context in it:

$$\begin{aligned} a(x) \mathbin{\overline{\circ}} \overline{b}c &:= (\tau/b = a), (\cdot[c/x] \mid \cdot) \\ a(x) \mathbin{\overline{\circ}} \overline{b}(c) &:= (\tau/b = a), (\nu c)(\cdot[c/x] \mid \cdot) \\ \overline{a}c \mathbin{\overline{\circ}} b(x) &:= (\tau/a = b), (\cdot \mid \cdot[c/x]) \\ \overline{a}(c) \mathbin{\overline{\circ}} b(x) &:= (\tau/a = b), (\nu c)(\cdot \mid \cdot[c/x]) \end{aligned}$$

$\overline{\circ}$ is not defined on any other kind of arguments.

We will work a lot with name substitution, so we will use the open bisimilarity:

Definition 5.1 (Open bisimilarity)**.** A symmetric relation R is an *open bisimulation* if for all name substitution σ, whenever $P \mathcal{R} Q$ and $P\sigma \xrightarrow{\mu} P'$, there exists Q' such that $Q\sigma \xrightarrow{\mu} Q'$ and $P' \mathcal{R} Q'$. Two processes P and Q are *open bisimilar*, written $P \sim_o Q$, if $P \mathcal{R} Q$ for some open bisimulation $\mathcal{R}$.

5.2 Behavioral laws and translation into πwI

This calculus does not automatically inherit the properties of the π-calculus, since it is a different transition system.

Lemma 5.1 ($\equiv \subset \sim_o$)**.** For all processes P, Q of πw, if P and Q are structurally equivalent then P and Q are open bisimilar.

Proof in appendix A.3. □

Lemma 5.2 ($\sim_o$ is a congruence)**.** For all C context of πw, if $P \sim_o Q$ then $C[P] \sim_o C[Q]$.

Proof. By induction on C, the only difficult case if $C = [\cdot] \mid R$, since the only non trivial rule to apply is *comp*, we only have to validate it under the action of $\mathcal{C}$ and fortunately we work up to substitution with the open bisimilarity. □

Lemma 5.3 (Transitivity)**.** For all $a, c \neq b$: $\{a = c\} \sim_o (\nu b)(\{a = b\} \mid \{b = c\})$

Proof. Straightforward: there is only one transition for each side (into the same processes). □

Lemma 5.4 (Substitution law)**.** For all $a \neq b$, even if b is not fresh:

$$P[b/a] \sim (\nu a)(\{a = b\} \mid P)$$

Proof. The corresponding relation is a strong bisimulation, there is no need to investigate P, only its transitions[4]. We can then reduce the problem to proving that $\beta\bullet(E+(a = b))\backslash\{a\} = (\beta\bullet E)[b/a]$, which is true. □

[4]Except for proving that if $P[b/a] \xrightarrow{\mu_1} P_1$ then $\exists \mu, P' \;\; P \xrightarrow{\mu} P' \;\wedge\; \mu_1 = \mu[b/a] \;\wedge P_1 = P'[b/a]$

Definition 5.2 (From πw to πIw)**.** The translation from πw into πIw is transparent on every constructor but the free output, so we omit all the other cases.

$$[\![\overline{a}b.P]\!]^{\pi wI}_{\pi w} = \overline{a}(b').(\{b' = b\} \mid [\![P]\!]^{\pi wI}_{\pi w})$$

Lemma 5.5 (Full abstraction*)**.**

$$P \sim_o Q \Leftrightarrow [\![P]\!] \sim_o [\![Q]\!]$$

Proof. Work in progress. Left to right is easy, but right to left seems to need other bisimilarities. □

Definition 5.3. The *dual* $\overline{P}$ of a process P of πIw is the same process with $a(x)$ replaced by $\overline{a}(x)$ and vice versa. There is no modification for wires. The *dual* $\overline{\mu}$ of an action μ is defined the same way. (but the dual of $\overline{a}b, E$ is not defined)

Fact 5.1. $\overline{\overline{P}} = P \quad ; \quad \overline{\overline{\mu}} = \mu \quad ; \quad P \xrightarrow{\mu} P' \Leftrightarrow \overline{P} \xrightarrow{\overline{\mu}} \overline{P'}$

Then the only difference between $\overline{[\![[\![\cdot]\!]^{\mathcal{M}_S}_p]\!]^{\pi wI}_{\pi w}}$ and $[\![[\![\cdot]\!]^{\mathcal{O}}_p]\!]^{\pi wI}_{\pi w}$ is now the difference between $[\![a(x).\overline{b}x]\!]^{\pi wI}_{\pi w} = a(x).\overline{b}(y).\{y = x\}$ and $\{a = b\}$, which is not much in presence of some $!\overline{a}(z)$ in the context. However this remains a work in progress and the duality between $[\![\cdot]\!]^{\mathcal{M}_S}_p$ and $[\![\cdot]\!]^{\mathcal{O}}_p$ is not completely clear here either.

6 Conclusion

The duality in π must be treated carefully. The fragment of π that were translatable into πI before (ALπ) cannot be dualized. The translation we introduced from a bigger fragment, π_{io} can be dualized but with some gaps, since we must handle the links processes with precautions.

We have introduced the calculus πw in the last part to be able to have a calculus with an LTS that has first-class operators, called wires, instead of links. The theory of πw is very new and is developped in this report. In this calculus all laws we wanted in πI are true and in a much easier way. The laws for the substitution and full abstraction are about strong bisimilarities which are closer relation. The duality is much easier to manipulate in πwI.

6.1 Future work

We should be able to complete the proofs of the substitution lemma (lemma 3.3) which seems to be true but all techniques we tried for now failed. However the proof of the duality in πI with links (lemma 4.2) is more fragile, since it raises problems mainly about the dual processes of the links, that do not raise the expected execution traces. This is the initial reason for the introduction of wires, that are self-dual. Also we plan to soon establish the duality of the encodings into πwI.

Once we complete and understand the theories of πI with links and πw we should relate it to existing works:

Building links makes us wonder if it is possible to build links for variables with both input and output capability, with type $\#T$. Equators [Mer99] juxtapose both version of links and are a partial solution to this problem, but they introduce divergences.

We will study the theory of the calculus πw we introduced in the last part without specifically focusing on the duality. The mechanism of joining names on one side and asking for joining names on the other side seems familiar and we could try to relate it to existing systems, like the fusion

calculus [PV98] where communication between processes can create connection instead of asking for them like in πw.

πwI is interesting for its duality, since it is probably more than just πI. However πwI seems to be interesting in itself for example to understand the mechanisms of name aliasing in a calculus with only bound outputs. We will also compare πwI with π.

Links similar to the ones we defined in sect. 2.2 are used in [BHY05] for channels with output capability only – where input capability corresponds to "universal types". Since the typing system is different we could try to adapt our links to it.

[GLW03] uses linear forwarders (first-class operators implementing the finite links) to deal with the input capability. The approach is more about the position of the links than about adapting the definition of the links to its current usage. It may be interesting to adapt this method to translate into πI.

Finally we can try to relate the existing encodings of λ with other evaluation strategies or with some less standard encodings built in [Vas05]. They might be more suitable to some translation into πI without heavy processes like the links. This would help to answer the question of the expressiveness of πI, at least regarding the Turing-completeness.

References

[BHY05] Martin Berger, Kohei Honda, and Nobuko Yoshida. Genericity and the pi-calculus. *Acta Inf.*, 42(2-3):83–141, 2005.

[Bor98] Michele Boreale. On the expressiveness of internal mobility in name-passing calculi. *Theoretical Computer Science*, 195:205–226, March 1998.

[GLW03] Philippa Gardner, Cosimo Laneve, and Lucian Wischik. Linear forwarders. In R. Amadio and D. Lugiez, editors, *CONCUR 2003*, volume 2761 of *Lecture Notes in Computer Science*, pages 415–430. Springer-Verlag, 2003.

[Mer99] Massimo Merro. On equators in asynchronous name-passing calculi without matching. *Electr. Notes Theor. Comput. Sci.*, 27:57–70, 1999.

[Mil92] Robin Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[MS04] Massimo Merro and Davide Sangiorgi. On asynchrony in name-passing calculi. *Mathematical Structures in Computer Science*, 14(5):715–767, 2004.

[PS96] Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996.

[PV98] Joachim Parrow and Björn Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *LICS*, pages 176–185, 1998.

[San96] Davide Sangiorgi. π-calculus, internal mobility and agent-passing calculi. *Theoretical Computer Science*, 167(2):235–274, 1996.

[SW01] Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes.* Cambridge University Press, 2001.

[Vas05] Vasco T. Vasconcelos. Lambda and pi calculi, cam and secd machines. 15(1):101–127, 2005.

[vBV09] Steffen van Bakel and Maria Grazia Vigliotti. A logical interpretation of the λ-calculus into the π-calculus, preserving spine reduction and types. In *CONCUR*, pages 84–98, 2009.

[vBV10] Steffen van Bakel and Maria Grazia Vigliotti. Implicative logic based encoding of the λ-calculus into the π-calculus, 2010.

Appendices

A.1 Calculi and subcalculi

The following subcalculi of π (or πw) are used in this report. All of them have the subject reduction property.

π: the π-calculus, with delayed inputs and usual, non-delayed, inputs, replicated prefixes

$A\pi$: asynchronous π, i.e. in $\overline{a}b.P$, P must be 0

$L\pi$: localized π, i.e. in $a(x).P$, x is not used in input subject position in P

$AL\pi$: both asynchronous and localized, ALπ=A$\pi \cap L\pi$.

πI: π with internal mobility only: $\overline{a}b.P$ is preceded by νb, and also with links (which are a special case than full recursive definitions)

π_{io}: π typed with the π_{io} type system, where only one capability is transmitted in messages.

πw: the π-calculus with wires

πwI: the π-calculus with wires with internal mobility only: $\overline{a}b.P$ is preceded by νb

A.2 Details on infinite links

The key point to understand the behavior of the link can be seen in the following interaction:

$$[\![\overline{a}b \mid a(x).P]\!]^I_{AL} \xrightarrow{\tau} (\nu x)(x \hookrightarrow b \mid [\![P]\!]^I_{AL})$$

Informally, after this communication every time x is used, there will be a bound output followed by a link and that second link will compose[5] with either $x \hookrightarrow b$ or a sublink[6] of $x \hookrightarrow b$. Namely, since P is in ALπ we know that:

[5] Thanks to the transitivity law: $(\nu x)(a \hookrightarrow x \mid x \hookrightarrow b) \approx a \hookrightarrow b$

[6] A link has some sub-processes that are themselves links, e.g. $y \overset{T}{\hookrightarrow} x$ is a sublink of $a \overset{oT}{\hookrightarrow} b = a(x).\overline{b}(y).y \overset{T}{\hookrightarrow} x$

- either x is in output subject, in that case
  $(\nu x)(x \hookrightarrow b \mid [\![\overline{x}c]\!]^I_{AL}) = (\nu x)(!x(u).\overline{b}(v).v \hookrightarrow u \mid \overline{x}(y).y \hookrightarrow c)$
  has no choice but to internally communicate into:
  $(\nu yx)(x \hookrightarrow b \mid \overline{b}(v).v \hookrightarrow y \mid y \hookrightarrow c)$
  $\sim \quad \overline{b}(v).(\nu y)(v \hookrightarrow y \mid y \hookrightarrow c)$
  $\approx \quad \overline{b}(v).v \hookrightarrow c = [\![\overline{b}c]\!]^I_{AL}$.

  In this case the sublink $v \hookrightarrow u$ of $x \hookrightarrow b$ will communicate with the link contained in $[\![\overline{x}c]\!]$, and this interaction will act like $[\![\overline{x}b]\!]$.

- or x is in output object, in that case
  $(\nu x)(x \hookrightarrow b \mid [\![\overline{c}x]\!]^I_{AL}) = (\nu x)(x \hookrightarrow b \mid \overline{c}(y).y \hookrightarrow x)$
  $\sim \quad \overline{c}(y).(\nu x)(y \hookrightarrow x \mid x \hookrightarrow b)$
  $\approx \quad \overline{c}(y).y \hookrightarrow b = [\![\overline{c}b]\!]^I_{AL}$

  In this case $x \hookrightarrow b$ will communicate with the link contained in $[\![\overline{c}x]\!]$ and this interaction will act like $[\![\overline{b}x]\!]$.

In either case, the link or one of its derivatives composes with another link inside the translation of an output. Links stay in the translated processes forever, interacting each time their head name is involved. This interaction take more time, in terms of transitions, than the original processes.

A.3 Proofs

Proof of lemma 2.1 (Subject reduction). Statement: If $\Gamma \vdash P$ and $P \xrightarrow{\mu} P'$ then $\Gamma' \vdash P'$ (with $\mathrm{dom}(\Gamma') = \mathrm{dom}(\Gamma) + \mathrm{bn}(\mu)$)
By induction on the derivation of $P \xrightarrow{\mu} P'$, we use a stronger induction hypothesis and tiresome lemmas that are mainly about subtyping.

- restriction to free names: if $\Gamma \vdash P$ then $\Gamma_{|\mathrm{fn}(P)} \vdash P$
- weakening: if $\Gamma \vdash P$ then $\Gamma, \Gamma' \vdash P$ for all Γ' ranging on fresh names
- substitution with smaller types: if $\Gamma, x : T \vdash P$ and $\Gamma \vdash b : T$ then $\Gamma \vdash P[b/x]$
- typing of the label: if $\Gamma \vdash P$ and $P \xrightarrow{\overline{a}b}$ then there is a strict type T such that $\Gamma \vdash a : oT, b : T$
- stronger induction hypothesis: $\Gamma \vdash P$ and $P \xrightarrow{\mu} P'$ then $\Gamma, \mathrm{bn}(\mu) : T \vdash P'$ where T is the type (or tuple of types) remaining when removing the top-level connective of the type of $\mathrm{fn}(\mu)$ (indeed $|\mathrm{fn}(\mu)| = 1$ when $\mathrm{bn}(\mu) \neq \emptyset$)

□

Proof of lemma 2.2 ($L\pi \subset \pi_{io}$). Statement: if P is a process of Lπ, then P is typable in π_{io} with the type $\#o^\omega$, where $o^\omega := \mu T.oT$:

$$\mathrm{fn}(P) : \#o^\omega \vdash P$$

We prove by induction on $P \in \mathrm{L}\pi$ that, for all V and N, if

1. $\mathrm{fn}(P) \subseteq N \uplus V$

2. $\forall v \in V$, v is not in input subject position in P

then $\quad N : \#o^\omega, V : o^\omega \vdash P$ □

Proof of lemma 3.1 (transitivity law). Statement: If $b \neq a, c$ then

$$\forall T \; a \overset{T}{\hookrightarrow} c \lesssim (\nu b)(a \overset{T}{\hookrightarrow} b \mid b \overset{T}{\hookrightarrow} c)$$

We build a relation $\mathcal{R}$ such that, if $P \; \mathcal{R} \; Q$, then:

$$\begin{aligned}
(P \mid a \overset{T}{\hookrightarrow} c) \;\; &\mathcal{R} \;\; (Q \mid (\nu b)(a \overset{T}{\hookrightarrow} b \mid b \overset{T}{\hookrightarrow} c)) \\
(P \mid \overline{c}(\tilde{z}).\tilde{z} \overset{\tilde{T}}{\hookrightarrow} \tilde{x}) \;\; &\mathcal{R} \;\; (Q \mid (\nu b)(\overline{b}(\tilde{y}).\tilde{y} \overset{\tilde{T}}{\hookrightarrow} \tilde{x} \mid b(\tilde{y}).\overline{c}(\tilde{z}).\tilde{z} \overset{\tilde{T}}{\hookrightarrow} \tilde{y})) \\
(P \mid c(\tilde{z}).\tilde{z} \overset{\tilde{T}}{\hookrightarrow} \tilde{x}) \;\; &\mathcal{R} \;\; (Q \mid (\nu b)(b(\tilde{y}).\tilde{y} \overset{\tilde{T}}{\hookrightarrow} \tilde{x} \mid \overline{b}(\tilde{y}).c(\tilde{z}).\tilde{z} \overset{\tilde{T}}{\hookrightarrow} \tilde{y})) \\
(P \mid \overline{c}(\tilde{z}).\tilde{z} \overset{\tilde{T}}{\hookrightarrow} \tilde{x}) \;\; &\mathcal{R} \;\; (Q \mid \overline{c}(\tilde{z}).(\nu \tilde{y})(\tilde{z} \overset{T}{\hookrightarrow} \tilde{y} \mid \tilde{y} \overset{\tilde{T}}{\hookrightarrow} \tilde{x})) \\
(P \mid c(\tilde{z}).\tilde{z} \overset{\tilde{T}}{\hookrightarrow} \tilde{x}) \;\; &\mathcal{R} \;\; (Q \mid c(\tilde{z}).(\nu \tilde{y})(\tilde{z} \overset{T}{\hookrightarrow} \tilde{y} \mid \tilde{y} \overset{\tilde{T}}{\hookrightarrow} \tilde{x}))
\end{aligned}$$

with pairwise distinct 'c's and 'a's, and no '$\tilde{x}$' in the 'a's. This way no communication is possible but those immediately preceded by (νb) or $(\nu \tilde{y})$. We show that on each side the subprocess can only do one transition to a parallel composition of zero or more subprocesses on the same side, and the corresponding process on the other side can do the same transition to the corresponding process on the other side, except for τ-transitions on the right which correspond to no transition on the left.

The relation $\mathcal{R}$ is an expansion relation up to structural congruence. □

Proof of lemma 3.2 (Replication law). Statement: If

1. a is the subject of the prefix α
2. no derivative of P_1 can communicate with a derivative of P_2 about[7] a
3. no derivative of R can make an action on a

then

$$(\nu a)(\alpha.R \mid P_1 \mid P_2) \sim (\nu a)(\alpha.R \mid P_1) \mid (\nu a)(\alpha.R \mid P_2)$$

The corresponding relation $\mathcal{R}$ (for all P_1, P_2) is a strong bisimulation up to ν-contexts.

- **action from** P_i: same action, same restriction on both sides. (side-conditions OK thanks to hyp. 2)
- **comm. between** P_i **and** $\alpha.R$: same action, the new P_i is now $(R \mid P_i')$ or $(\nu x)(R \mid P_i')$ (side-conditions OK thanks to hyps. 2 and 3)
- **comm. between** P_1 **and** P_2**, right to left**: same action. It can generate a extra (νx), handled by the up to ν-contexts and up to $\equiv$.

[7] i.e. with actions neither of subject a nor of object a

- **comm. between P_1 and P_2, left to right**: same as above, and the hyp. 2 make (νa) harmless.

□

Lemma A.1 (Misc. laws)**.**

- $(\nu ax)(a \overset{T}{\hookrightarrow} b \mid b(y).x \overset{T_1}{\hookrightarrow} y \mid Q) \lesssim (\nu a)(a \overset{T}{\hookrightarrow} b \mid a(x) : Q)$
- if x the subject of α then, and $T = oT_1$ is the type of a, then:

$$(\nu az)(a \overset{T}{\hookrightarrow} b \mid \overline{b}(y).z \overset{T_1}{\hookrightarrow} z \mid (\alpha.P)[z/x]) \lesssim (\nu a)(a \overset{T}{\hookrightarrow} b \mid \overline{a}(x).\alpha.P)$$

- $a(x) : (P \mid Q) \sim P \mid a(x) : Q$ if $x \notin fn(P)$
- $a(x) : (\nu c)P \sim (\nu c)(a(x) : P)$ if $a \neq c$

Proof of lemma 3.3. . Statement: If a is substitutable in P, let T be any type for which there exists Γ such that $\Gamma, a : T \vdash P$. Then:

$$[\![P]\!]^I_{io}[b/a] \lesssim (\nu a)(a \overset{T}{\hookrightarrow} b \mid [\![P]\!]^I_{io})$$

By induction on the size of P, we prove that:
if $(\Gamma, \tilde{a} : \tilde{T} \vdash P)$ and ($\tilde{a}$ are substitutable in P) then

$$[\![P]\!][\tilde{b}/\tilde{a}] \lesssim (\nu \tilde{a})(\tilde{a} \overset{\tilde{T}}{\hookrightarrow} \tilde{b} \mid [\![P]\!]) \tag{3}$$

$$b(x) : [\![P]\!] \lesssim (\nu \tilde{a})(b(y).x \overset{T}{\hookrightarrow} y \mid [\![P]\!]) \text{ if } x \in \tilde{a} \tag{4}$$

Case analysis on P, for (3):

- $P = 0$: both sides are ~ 0
- $P = (\nu c)P_1$: induction hypothesis and static contexts
- $P = P_1 \mid P_2$: induction hypotheses, static contexts and replication law.
- $P = c(z).P_1$ or $P = \overline{c}d.P_1$ with $c \notin \tilde{a}$: induction hypothesis and deterministic transition
- $P = !c(z).P_1$ with $c \notin \tilde{a}$: induction hypothesis and deterministic relation up to $(|/ \lesssim)$-contexts and $\sim$.
- $P = c(z) : P_1$ with $c \notin \tilde{a}$: induction hypothesis and lemma A.1
- $P = a(x).P_1$, $P = !a(x).P_1$ or $P = \overline{a}c.P_1$ (with $a \in \tilde{a}$): does not happen, since a is substitutable
- $P = a(x) : P_1$ with $a \in \tilde{a}$: induction hypotheses and lemma A.1
- $P = \overline{a}c$ with $a \in \tilde{a}$: lemma A.1 and transitivity law
- $P = \overline{c}a.P_1$: deterministic transition, induction, replication law and transitivity law

Case (4):
We now have to prove (4) knowing ((4) for smaller Ps) and ((3) for P and smaller Ps).
(Work in progress: it is harder to exploit induction hypotheses)

□

Proof of lemma 5.1. Equivalent statement: for all P, Q, R processes of πw,

1. $P \mid 0 \sim_o P$
2. $P \mid Q \sim_o Q \mid P$
3. $P \mid (Q \mid R) \sim_o (P \mid Q) \mid R$
4. $(\nu a)(P \mid Q) \sim_o (\nu a)P \mid Q$ if $a \notin \mathrm{fn}(Q)$
5. $(\nu a)(\nu b)P \sim_o (\nu b)(\nu a)P$
6. $(\nu a)0 \sim_o 0$
7. $!P \sim_o P \mid !P$

1,5,6,7 are straightforward; 2 comes from the fact that the rules of the LTS are symmetric (thanks to $\circ$ symmetric), 4 needs careful checking and 3 uses 5 and 6 as well as the fact that the composition of labels $\circ$ is associative. □