Output Without Delay: A π-Calculus Compatible with Categorical Semantics

Ken Sakayori
The University of Tokyo, Japan

Takeshi Tsukada
Chiba University, Japan

Abstract

The quest for logical or categorical foundations of the π-calculus (not limited to session-typed variants) remains an important challenge. A categorical type theory correspondence for a variant of the **i**/**o**-typed π-calculus was recently revealed by Sakayori and Tsukada, but, at the same time, they exposed that this categorical semantics contradicts with most of the behavioural equivalences. This paper diagnoses the nature of this problem and attempts to fill the gap between categorical and operational semantics. We first identify the source of the problem to be the mismatch between the operational and categorical interpretation of a process called the forwarder. From the operational viewpoint, a forwarder may add an arbitrary delay when forwarding a message, whereas, from the categorical viewpoint, a forwarder must not add any delay when forwarding a message. Led by this observation, we introduce a calculus that can express forwarders that do not introduce delay. More specifically, the calculus we introduce is a variant of the π-calculus with a new operational semantics in which output actions are forced to happen as soon as they get unguarded. We show that this calculus (i) is compatible with the categorical semantics and (ii) can encode the standard π-calculus.

2012 ACM Subject Classification Theory of computation → Concurrency; Theory of computation → Process calculi; Theory of computation → Denotational semantics

Keywords and phrases π-calculus, categorical semantics, linear approximation

Digital Object Identifier 10.4230/LIPIcs.FSCD.2021.32

Funding This work was supported by JSPS KAKENHI Grant Numbers JP20J13473 and JP19K20211.

Acknowledgements We would like to thank anonymous referees for useful comments.

1 Introduction

The connection between the π-calculus and logic or categorical type theory has been studied since the early stages of the development of the π-calculus [1, 3, 2]. Among others, a close correspondence between a session typed π-calculus and intuitionistic linear logic [6] (and hence also the relationship to categorical models of linear logic) is well-understood. The session-typed calculi corresponding linear logic, however, are not quite expressive since they are race-free and deadlock-free. So it is natural to question whether a similar categorical foundation can be given to processes not limited to deadlock-free and race-free processes.

A fundamental difficulty in developing a categorical type theory for process calculi in the presence of race condition has been recently pointed out by Sakayori and Tsukada [19]. They showed that asynchronous π-calculus processes modulo observational equivalence (weak barbed congruence) do not form a category, under some mild assumptions [19, Theorem 1].[1] Hence, the observational equivalence cannot be an instance of an equational theory characterised by a certain categorical structure; this is in contrast to the case of λ-calculus, where observational equivalence is a $\beta\eta$-theory.

[1] The choice of the behavioural equivalence does not matter since their argument also applies to many other behavioural equivalences, such as must-testing equivalence.

Hence, if a process calculus based on the (asynchronous) π-calculus were to have some categorical foundation, its operational behaviour must be distant from conventional behaviour.

This paper introduces a variant of the π-calculus whose observational equivalence harmonises with categorical semantics. We introduce a novel reduction semantics to the π-calculus and show that processes modulo weak barbed congruence, defined on top of the new reduction semantics, form a category; more precisely, they form a compact closed Freyd category [19] (described below).

Before introducing the operational semantics that we propose, let us explain the problem of conventional behavioural equivalences in a little more detail.

The problem is about the behaviour of a special process $!a(x).\bar{b}\langle x\rangle$, which is often called a *forwarder* or a *link*. Intuitively this process transfers a message from channel a to channel $\bar{b}$. This intuition justifies the following equation

$$(\boldsymbol{\nu} a)(P \mid !a(x).\bar{b}\langle x\rangle) \qu= \quad P\{\bar{b}/\bar{a}\}, \qquad a, \bar{b} \notin \mathbf{fn}(P), \tag{1}$$

which indeed holds for the weak barbed congruence in an asynchronous setting. If we adopt "parallel composition + hiding" as the notion of composition, i.e. if we regard $(\boldsymbol{\nu} a)(P \mid !a(x).\bar{b}\langle x\rangle)$ as a composition of $!a(x).\bar{b}\langle x\rangle$ and P, (1) says that the forwarder is a right-identity. If processes modulo weak barbed congruence formed a category, a right-identity would be the identity and in particular a left-identity, as in any other categories. The left-identity law is

$$(\boldsymbol{\nu} b)(!a(x).\bar{b}\langle x\rangle \mid P) \quad = \quad P\{a/b\}, \qquad a, \bar{b} \notin \mathbf{fn}(P), \tag{2}$$

but this is invalid with respect to weak barbed congruence.

To see why (2) fails for weak barbed congruence, let us review the conventional behavioural interpretation of a forwarder $!a(x).\bar{b}\langle x\rangle$: it receives a message from a, *possibly waits as long as it wants or needs*, and then sends the message to the receiver. Hence the process $(\boldsymbol{\nu} b)(!a(x).\bar{b}\langle x\rangle \mid P)$ can immediately receive a message from a and keep it until P actually requires a message from b. On the other hand, $P\{a/b\}$ do not receive a message from a unless $P\{a/b\}$ actually requires it. This difference is significant in the presence of race condition, and thus (2) fails for weak barbed congruence.

A similar observation on a problem caused by delays introduced by forwarders and a solution against that problem has been made in the context of game semantics. When giving a game semantics of a synchronous session typed π-calculus, Castellan and Yoshida [7] observed that the (traditional) copycat strategy – the game semantic counterpart of the forwarder process – does not behave as identity due to the delay it introduces. To avoid this problem, they introduced a copycat strategy that does not introduce any delay and proved that this "delayless copycat strategy" works as the identity.

Whereas [7] added delayless forwarders as semantic elements that processes cannot represent, this paper discusses a new operational semantics on processes with respect to which forwarders are delayless. The main result shows that behavioural equivalences under the delayless interpretation is in harmony with categorical semantics.

The new operational semantics introduced in this paper is a reduction semantics that forces output actions to happen as soon as they get unguarded. Under the new operational semantics, when a forwarder $!a(x).\bar{b}\langle x\rangle$ receives a message m from a, it *must immediately send* m to a receiver b. In other words, the following two transitions are atomic

$$!a(x).\bar{b}\langle x\rangle \quad \xrightarrow{a(m)} \quad \underline{!a(x).\bar{b}\langle x\rangle \mid \bar{b}\langle m\rangle} \quad \xrightarrow{\bar{b}\langle m\rangle} \quad !a(x).\bar{b}\langle x\rangle,$$

and the process cannot stop at the underlined intermediate step since it has an unguarded output action. So one-step reduction in our calculus corresponds to multi-step reduction in the conventional calculus. We may consider that the new behaviour expresses a synchronous communication since a message m now cannot be kept in a communication medium $\bar{a}\langle m\rangle$.

In our proposed calculus, processes modulo observational equivalence form a compact closed Freyd category. This means that not only equations (1) and (2) but also some equational laws studied for the asynchronous π-calculus are valid under this new operational behaviour. This is because compact closed Freyd category is a categorical structure that corresponds to a theory of processes, i.e. a congruence over asynchronous π-processes satisfying certain equational laws [19]. One of the laws is (2), and the others are laws that frequently appear in the study of asynchronous π-calculus, such as the replication theorem [17].

We also show that a π-calculus with the standard reduction semantics, can be embedded into the proposed calculus by using a special constant τ for delay. The translation replaces each output action $\bar{a}\langle m\rangle$ with $\tau.\bar{a}\langle m\rangle$, making explicit the delay of the output action in the conventional π-calculus. For instance, the conventional behaviour of a forwarder $!a(x).\bar{b}\langle x\rangle \xrightarrow{a(m)} !a(x).\bar{b}\langle x\rangle \mid \bar{b}\langle m\rangle$ is mimicked by $!a(x).\tau.\bar{b}\langle x\rangle \xrightarrow{a(m)} !a(x).\bar{b}\langle x\rangle \mid \tau.\bar{b}\langle m\rangle$ in the new operational semantics.

Technically the new operational semantics is quite complicated since its one-step reduction is a multi-step reduction with a certain condition in the conventional calculus. To overcome the difficulty in reasoning about such a complicated calculus, we develop an intersection type system, or equivalently a system of linear approximations [21, 14], that captures the behaviour of a process. We think that the system would be of independent technical interest.

Organisation of the paper

Section 2 introduces our calculus and states the main result; the following sections are devoted to its proof. After reviewing the idea of linear approximations and its correspondence to reduction sequences in Section 3, we formalise this idea in Section 4. Section 5 defines an LTS based on linear approximations, and Section 6 shows that barbed congruence has a categorical model. Section 7 discusses related work and Section 8 concludes the paper.

2 A process calculus with undelayed output

This section (i) introduces a variant of the π-calculus whose barbed congruence can be captured categorically and (ii) claims the main result of this paper. The syntax of the calculus is the same as that of the π_F-calculus introduced by Sakayori and Tsukada [19], but the calculus is equipped with a non-standard reduction semantics; *we also call this calculus the π_F-calculus.*[2] The proof of the main result will be given in the following sections.

2.1 Syntax

The π_F-calculus is a variant of the polyadic asynchronous π-calculus with **i**/**o**-types,[3] which this paper calls *sorts* in order to avoid confusion with intersection types introduced later.

▶ **Definition 1** (Sorts). *The set of* sorts*, ranged over by S and T, is given by*

$$S, T ::= \mathbf{ch}^o[T_1, \ldots, T_n] \mid \mathbf{ch}^i[T_1, \ldots, T_n] \qquad (n \geq 0).$$

The sort $\mathbf{ch}^o[T_1, \ldots, T_n]$ (resp. $\mathbf{ch}^i[T_1, \ldots, T_n]$) is for channels for sending (resp. receiving) n arguments of types $T_1, \ldots, T_n$. We often write $\vec{T}$ for a sequence of sorts $T_1, \ldots, T_n$. The dual *$T^\perp$ of sort T is defined by $\mathbf{ch}^o[\vec{T}]^\perp \stackrel{\text{def}}{=} \mathbf{ch}^i[\vec{T}]$ and $\mathbf{ch}^i[\vec{T}]^\perp \stackrel{\text{def}}{=} \mathbf{ch}^o[\vec{T}]$.*

[2] Although the π_F-calculus introduced in this paper and the original π_F-calculus [19] have different reduction semantics it is not that odd to call them with the same name. This is because the reduction semantics is not essential to establish the correspondence to compact closed Freyd categories; we only need the "algebraic semantics" to establish the correspondence (cf. Appendix A).

[3] Unlike the original **i**/**o**-types [17], no names have both input and output capabilities. Names are used to represent the input/output endpoints of a channel.

$$\frac{\Delta \vdash P \qquad \Delta \vdash Q}{\Delta \vdash P \mid Q} \qquad \frac{\Delta, x : T, y : T^{\perp} \vdash P}{\Delta \vdash (\boldsymbol{\nu}_T xy)P} \qquad \frac{(x : \mathbf{ch}^i[\vec{T}]) \in \Delta \quad \Delta, \vec{y} : \vec{T} \vdash P}{\Delta \vdash !x(\vec{y}).P}$$

$$\frac{(x : \mathbf{ch}^o[\vec{T}]) \in \Delta \quad \vec{y} : \vec{T} \subseteq \Delta}{\Delta \vdash x\langle\vec{y}\rangle} \qquad \frac{(\tau : \mathbf{ch}^i[]) \in \Delta \quad \Delta \vdash P}{\Delta \vdash \tau.P} \qquad \frac{}{\Delta \vdash \mathbf{0}}$$

Figure 1 Sort assignment rules for processes.

▶ **Definition 2** (Processes). *The set of* processes *is defined by*

$$P, Q, R ::= \mathbf{0} \mid (P|Q) \mid (\boldsymbol{\nu}_T xy)P \mid x\langle\vec{y}\rangle \mid !x(\vec{y}).P \mid \tau.P,$$

where x and y range over a set of names and $\vec{y}$ represents a (possibly empty) sequence of names. We often elide sort annotations and write $(\boldsymbol{\nu}xy)$ for $(\boldsymbol{\nu}_T xy)$. The set of free names of P, written $\mathbf{fn}(P)$, and bound names of P written $\mathbf{bn}(P)$ are defined as usual.

All the constructs, except for the name restriction, are standard so their meaning should be clear.[4] The name restriction $(\boldsymbol{\nu}_T xy)P$ hides the names x and y of type T and $T^{\perp}$ and, at the same time, establishes a connection between x and y. The input-output connection is *not a priori* and communications only happen over bound names connected by $\boldsymbol{\nu}$; this is different from the standard π-calculi where $\bar{a}$ is considered as an output to a.

For a technical reason, we introduce not only *structural congruence*, but also a notion called *structural precongruence* $\Rrightarrow$ (cf. Remark 10). A precongruence is like a congruence, but it is just reflexive and transitive, not necessarily symmetric. We define $\Rrightarrow$ as the smallest precongruence relation on processes that satisfies the following rules:

$$P \mid \mathbf{0} \Lleftrightarrow P \qquad P \mid Q \Lleftrightarrow Q \mid P \qquad (P \mid Q) \mid R \Lleftrightarrow P \mid (Q \mid R)$$
$$(\boldsymbol{\nu}wx)(\boldsymbol{\nu}yz)P \Lleftrightarrow (\boldsymbol{\nu}yz)(\boldsymbol{\nu}wx)P \qquad ((\boldsymbol{\nu}xy)P) \mid Q \Rrightarrow (\boldsymbol{\nu}xy)(P \mid Q)$$

where $P \Lleftrightarrow Q$ means $P \Rrightarrow Q$ and $Q \Rrightarrow P$, w, x, y, z are distinct in the fourth rule and $x, y \notin \mathbf{fn}(Q)$ in the fifth rule. Unlike the structural congruence, the restriction of the scope of $(\boldsymbol{\nu}xy)$ is not allowed. The *structural congruence* $\equiv$ is the symmetric closure of $\Rrightarrow$.

The typing rules are rather straightforward. A *sort environment*, written Δ, is a finite set of bindings of the form $t\colon T$, where t is either a name x or τ, such that the names in Δ are pairwise distinct. The *sort assignment relation* $\Delta \vdash P$ is the least relation closed under the rules listed in Figure 1.

2.2 Reduction semantics

As mentioned in Section 1, a one-step reduction in our calculus corresponds to a multi-step reduction in the conventional calculus. So we first introduce the conventional reduction relation $\rightarrow$ and then define a new reduction relation $\Rightarrow$ using the conventional reduction.

The *standard reduction relation* $\xrightarrow{\ell}$ ($\ell = \tau$ or 0) is defined by the base rules

$$(\boldsymbol{\nu}\vec{w}\vec{z})(\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid \bar{a}\langle\vec{y}\rangle \mid Q) \xrightarrow{0} (\boldsymbol{\nu}\vec{w}\vec{z})(\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid P\{\vec{y}/\vec{x}\} \mid Q)$$
$$(\boldsymbol{\nu}\vec{w}\vec{z})(\tau.P \mid Q) \xrightarrow{\tau} (\boldsymbol{\nu}\vec{w}\vec{z})(P \mid Q)$$

[4] Another notable characteristic of the π_F-calculus is that it does not have non-replicated inputs $a(\vec{x}).P$.

together with the structural rule which concludes $P \xrightarrow{\ell} Q$ from $P \Rrightarrow P' \xrightarrow{\ell} Q' \Rrightarrow Q$ for some P' and Q'. We write $P \to Q$ if the label is not important. The following is an example of a (multi-step) reduction:

$$\begin{aligned}
& (\boldsymbol{\nu}\bar{a}a)(\boldsymbol{\nu}\bar{b}b)(\tau.\bar{a}\langle m\rangle \mid !a(x).\bar{b}\langle x\rangle \mid !b(y).!c(z).P) \\
\xrightarrow{\tau}\ & (\boldsymbol{\nu}\bar{a}a)(\boldsymbol{\nu}\bar{b}b)(\bar{a}\langle m\rangle \mid !a(x).\bar{b}\langle x\rangle \mid !b(y).!c(z).P) \\
\xrightarrow{0}\ & (\boldsymbol{\nu}\bar{a}a)(\boldsymbol{\nu}\bar{b}b)(!a(x).\bar{b}\langle x\rangle \mid \bar{b}\langle m\rangle \mid !b(y).!c(z).P) \\
\xrightarrow{0}\ & (\boldsymbol{\nu}\bar{a}a)(\boldsymbol{\nu}\bar{b}b)(!a(x).\bar{b}\langle x\rangle \mid !c(z).P\{m/y\} \mid !b(y).!c(z).P).
\end{aligned}$$

In our calculus, the output action $\bar{b}\langle x\rangle$ in $!a(x).\bar{b}\langle x\rangle$ (resp. $\bar{a}\langle m\rangle$ in $\tau.\bar{a}\langle m\rangle$) must be performed at the same time as the input action $a(x)$ (resp. τ). Therefore, the above multi-step reduction should be regarded as a one-step reduction:

$$\begin{aligned}
& (\boldsymbol{\nu}\bar{a}a)(\boldsymbol{\nu}\bar{b}b)(\tau.\bar{a}\langle m\rangle \mid !a(x).\bar{b}\langle x\rangle \mid !b(y).!c(z).P) \\
\Rrightarrow\ & (\boldsymbol{\nu}\bar{a}a)(\boldsymbol{\nu}\bar{b}b)(!a(x).\bar{b}\langle x\rangle \mid !c(z).P\{m/y\} \mid !b(y).!c(z).P).
\end{aligned}$$

We formally define $\Rrightarrow$. A process P has an *unguarded output action* if $P \equiv (\boldsymbol{\nu}\vec{w}\vec{z})(\bar{a}\langle\vec{x}\rangle \mid Q)$ for some Q. A process with an unguarded output action is regarded as an incomplete, intermediate state that needs to perform further actions to complete an "atomic operation". We say that P is *settled* if P has no unguarded output action. We write $P \Rrightarrow Q$ if $P \xrightarrow{\tau}(\xrightarrow{0})^* Q$ and Q is settled.

The notion of *barbed congruence* can be easily adapted to this setting.

▶ **Definition 3** (Barbed bisimulation and barbed congruence)**.** *Let $\mathcal{R}$ be a binary relation on settled processes. We say that $\mathcal{R}$ is a* barbed bisimulation *if whenever $P\ \mathcal{R}\ Q$,*

1. *$P{\downarrow^{\tau}_{\bar{a}}}$ if and only if $Q{\downarrow^{\tau}_{\bar{a}}}$*
2. *$P \Rrightarrow P'$ implies $Q \Rrightarrow Q'$ and $P'\ \mathcal{R}\ Q'$ for some process Q'*
3. *$Q \Rrightarrow Q'$ implies $P \Rrightarrow P'$ and $P'\ \mathcal{R}\ Q'$ for some process P',*

where $P{\downarrow^{\tau}_{\bar{a}}}$ means that $P \xrightarrow{\tau} (\xrightarrow{0})^* \equiv (\boldsymbol{\nu}\vec{x}\vec{y})(\bar{a}\langle\vec{z}\rangle \mid P')$ and $\bar{a}$ is a free name of P.

The barbed bisimilarity $\overset{\bullet}{\sim}_{\tau}$ *is the largest barbed bisimulation. Processes P and Q are* barbed congruent*, written $P \simeq^{c}_{\tau} Q$, if $\tau.C[P] \overset{\bullet}{\sim} \tau.C[Q]$ for all context C. (The additional τ-prefixing is to ensure that the processes are settled.)*

The main result of this paper is that there exists a categorical model that is fully abstract with respect to $\simeq^{c}_{\tau}$. We use the categorical structure named *compact closed Freyd category* [19] to interpret π_F-calculus processes. The proof is given in the subsequent sections.

▶ **Theorem 4.** *π_F-processes modulo $\simeq^{c}_{\tau}$ forms a compact closed Freyd category. Hence there exists a compact closed Freyd category that is fully abstract with respect to $\simeq^{c}_{\tau}$.*

The proof can be easily adapted to prove a similar claim for any other congruence that subsumes $\simeq^{c}_{\tau}$, such as weak barbed congruence (for $\Rrightarrow$).

2.3 Relationship to the standard semantics

We have introduced two reduction relations to the π_F-calculus, namely $\to$ and $\Rrightarrow$. There exists an embedding of the π_F-calculus with $\to$ to that with $\Rrightarrow$.

The translation is quite simple: it replaces each output action $\bar{a}\langle\vec{x}\rangle$ with $\tau.\bar{a}\langle\vec{x}\rangle$, reflecting the fact that an output action in the standard semantics can be delayed. Let us write $(-)^{\dagger}$ for this translation. It preserves the semantics in the following sense.

▶ **Proposition 5.** *Suppose* $\Delta \vdash P$. *Then (i)* $P \to Q$ *implies* $(P)^\dagger \Rightarrow (Q)^\dagger$, *(ii)* $(P)^\dagger \Rightarrow Q'$ *implies* $Q' = (Q)^\dagger$ *and* $P \to Q$ *for some* Q, *and (iii)* $P{\downarrow_{\bar{a}}}$ *iff* $(P)^\dagger{\downarrow^{\tau}_{\bar{a}}}$.

From this proposition and the compositionality of $(-)^\dagger$, we obtain the following result. Let $\simeq^c$ for the conventional (strong) barbed congruence for π_F-processes, defined by replacing $\Rightarrow$ with $\longrightarrow$ and $\downarrow^{\tau}_{\bar{a}}$ with $\downarrow_{\bar{a}}$ (i.e. existence of a free unguarded output $\bar{a}$) in Definition 3.

▶ **Theorem 6.** *If* $\Delta \vdash P$, $\Delta \vdash Q$ *and* $(P)^\dagger \simeq^c_\tau (Q)^\dagger$, *then* $P \simeq^c Q$.

This translation, however, is *not* fully abstract with respect to barbed congruence. Contexts that are not in the image of the translation $(-)^\dagger$ give additional observational power.

3 Overview

To prove Theorem 4, we appeal to an axiomatic characterisation of compact closed Freyd category, proved in [19]: π_F-processes modulo an equivalence relation $\mathcal{R}$ forms a compact closed Freyd category if and only if $\mathcal{R}$ is a congruence satisfying six axioms[5], such as (2) and

$$(\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid C[\bar{a}\langle\vec{y}\rangle]) = (\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid C[P\{\vec{y}/\vec{x}\}]),\ a \notin \mathbf{fn}(P,C),\ \bar{a} \notin \mathbf{bn}(C), \tag{3}$$

where C is a context. Since barbed congruence is a congruence by definition, it suffices to check that barbed congruence satisfies the axioms.

However, checking the required axioms directly using the definition of $\Rightarrow$ in Section 2 does not seem tractable. Recall that $P \Rightarrow Q$ is indeed a reduction sequence $P \xrightarrow{\tau} P_1 \xrightarrow{0} \ldots \xrightarrow{0} P_n \xrightarrow{0} Q$. The problem is that $P_i \xrightarrow{0} P_{i+1}$ is defined in terms of the structure of P_i, which may be quite different from that of P. A representation of reduction sequence defined by structural induction on P, without directly referring to P_i, would be desirable.

We thus utilise the correspondence of (i) reduction sequences, (ii) derivations in an intersection type system, and (iii) linear approximations [21, 14].

An example of a linear approximation is $(a_1.\tau_1.\bar{a}_2 \parallel a_2.\bot) \sqsubset !a.\tau.\bar{a}$ where the green part is the linear approximation of the right-hand side. A linear approximation is *linear* in the sense that each name is used exactly once and all inputs are non-replicated; it is an *approximation* in the sense that some part is discarded (e.g. $\bot \sqsubset \tau.\bar{a}$ or $\bot \sqsubset !a.\tau.\bar{a}$) and replicated inputs are replaced by a finite number of its copies (e.g. $(a_1.\tau_1.\bar{a}_2 \parallel a_2.\bot) \sqsubset !a.\tau.\bar{a}$).[6]

To see how a linear approximation corresponds to a reduction sequence, let us consider the following linear approximation:

$$\begin{aligned}(\boldsymbol{\nu}[\langle\bar{a}_1,a_1\rangle\langle\bar{a}_2,a_2\rangle\langle\bar{a}_3,a_3\rangle])((a_1.\tau_1.(\bar{a}_2 \mid \bar{a}_3) \parallel a_2.\bot) \ \mid\ \tau_2.\bar{a}_1 \ \mid\ a_3.\bot)& \\ \sqsubset\quad (\boldsymbol{\nu}\bar{a}a)(!a.\tau.(\bar{a} \mid \bar{a}) \ \mid\ \tau.\bar{a} \ \mid\ !a.\tau.\bar{b}).&\end{aligned}$$

Because of linearity, a linear approximation is race-free; hence it induces an essentially unique reduction sequence. For example,

$$\begin{aligned}
&(\boldsymbol{\nu}[\langle\bar{a}_1,a_1\rangle\langle\bar{a}_2,a_2\rangle\langle\bar{a}_3,a_3\rangle])((a_1.\tau_1.(\bar{a}_2 \mid \bar{a}_3) \parallel a_2.\bot) \mid \tau_2.\bar{a}_1 \mid a_3.\bot) && (4)\\
\xrightarrow{\tau_2}\xrightarrow{0}\quad &(\boldsymbol{\nu}[\langle\bar{a}_2,a_2\rangle\langle\bar{a}_3,a_3\rangle])(a_2.\bot \mid \tau_1.(\bar{a}_2 \mid \bar{a}_3) \mid a_3.\bot)\\
\xrightarrow{\tau_1}\xrightarrow{0}\quad &(\boldsymbol{\nu}[\langle\bar{a}_2,a_2\rangle])(a_2.\bot \mid (\bar{a}_2 \mid \bot) \mid \bot) \quad\xrightarrow{0}\quad (\boldsymbol{\nu}[])(\bot \mid (\bot \mid \bot) \mid \bot).
\end{aligned}$$

[5] The axioms can be found in the Appendix A; please refer to [19] for a more detailed description.

[6] In the approximation, $\mid$ represents parallel-composition coming from the original process, whereas $p \parallel q$ means that p and q originate from the same replicated (sub)process.

Importantly a reduction sequence of an approximation canonically induces that of the approximated process: the reduction sequence corresponding to (4) is

$$\begin{aligned}(\boldsymbol{\nu}\bar{a}a)(!a.\tau.(\bar{a} \mid \bar{a}) \mid \tau.\bar{a} \mid !a.\tau.\bar{b}) &\xrightarrow{\tau}\xrightarrow{0} (\boldsymbol{\nu}\bar{a}a)(!a.\tau.(\bar{a} \mid \bar{a}) \mid \tau.(\bar{a} \mid \bar{a}) \mid !a.\tau.\bar{b}) \\ &\xrightarrow{\tau}\xrightarrow{0} (\boldsymbol{\nu}\bar{a}a)(!a.\tau.(\bar{a} \mid \bar{a}) \mid (\bar{a} \mid \tau.\bar{b}) \mid !a.\tau.\bar{b}) \\ &\xrightarrow{0} (\boldsymbol{\nu}\bar{a}a)(!a.\tau.(\bar{a} \mid \bar{a}) \mid (\tau.(\bar{a} \mid \bar{a}) \mid \tau.\bar{b}) \mid !a.\tau.\bar{b}).\end{aligned} \tag{5}$$

Via the three-way correspondence mentioned above, this phenomenon can be understood as Subject Reduction of the intersection type system.

Conversely, given a reduction sequence, we can construct a linear approximation that represents the reduction sequence. This is a consequence of Subject Expansion, namely, $p \sqsubset P$ and $Q \to P$ imply $q \sqsubset Q$ and $q \to p$ for some q. The approximation for $P_1 \to P_2 \to \ldots \to P_n$ is obtained by iteratively applying this lemma to $p_n \sqsubset P_n$, where p_n is the approximation that discards everything.

So far, we have discussed a relationship between $\{\, Q \mid P \to^* Q \,\}$ and $\{\, p \mid p \sqsubset P \,\}$. This relation can be seen as a bisimulation, by appropriately introducing a relation to $\{\, p \mid p \sqsubset P \,\}$. Note that such a relation is not the reduction, since $p \to q$ changes the subject, i.e. $q \sqsubset Q$ for some Q with $P \xrightarrow{0} Q$ but not $q \sqsubset P$. Instead, we introduce an "ordering" over approximations of P. The idea is that a longer reduction sequence corresponds to a larger approximation. We write $p_1 \trianglelefteq p_2$ if p_1 is obtained by discarding some (sub)processes of p_2. For example, the second step of (5) corresponds to

$$(\boldsymbol{\nu}[\langle \bar{a}_1, a_1\rangle])((a_1.\bot) \mid \tau_2.\bar{a}_1 \mid \bot) \trianglelefteq (\boldsymbol{\nu}[\langle \bar{a}_1, a_1\rangle\langle \bar{a}_3, a_3\rangle])(a_1.\tau_1.(\bot \mid \bar{a}_3) \mid \tau_2.\bar{a}_1 \mid a_3.\bot),$$

representing that the third process in (5) is obtained by performing the actions corresponding to τ_1 and $\bar{a}_3$.

The bisimilarity gives us a characterisation of the behaviour of a process P in terms of linear approximations (or intersection type derivations) for P. Then Theorem 4 can be proved by "proof manipulation". For example, the proof of the above mentioned axiom $(\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid C[\bar{a}\langle\vec{y}\rangle]) = (\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \mid C[P\{\vec{y}/\vec{x}\}])$ resembles to the proof of the substitution lemma in a typical type system.

4 Linear approximation and execution sequence

We introduce *linear processes* by which executions of processes can be described.

4.1 Linear processes and intersection types

We start by defining linear processes. Although the definition of linear processes depends on the definition of intersection types because processes are annotated by types, we defer defining types for the sake of presentation.

▶ **Definition 7** (Linear processes). *A* linear name *is an object of the form x_i where x is an ordinary name and i is a natural number. Similarly, a* linear term*, denoted by t, is either a linear name or a constant of the form τ_i.*

Linear processes *are defined by the following grammar:*

$$\begin{aligned} p, q &::= \mathbf{0} \mid x_i\langle\lambda_1, \ldots, \lambda_n\rangle \mid x_i(\mu_1, \ldots, \mu_n).p \mid \tau_i.p \\ &\quad \mid (p \mid q) \mid (p_1 \parallel \cdots \parallel p_n) \mid (\boldsymbol{\nu}[\langle x_{i_1}, y_{i_1}\rangle_{\rho_1}, \ldots, \langle x_{i_n}, y_{i_n}\rangle_{\rho_n}])p \\ \mu &::= \langle x_{i_1}, \ldots, x_{i_n}\rangle \qquad \lambda ::= \langle \varphi_1 \cdot x_{i_1}, \ldots, \varphi_n \cdot x_{i_n}\rangle \end{aligned}$$

Here name restriction is annotated with types ρ_i, and the argument of an output action is annotated with witnesses of type isomorphisms φ_i. (The notion of types and type isomorphisms are introduced below and thus can be ignored for the moment.) In the above definition n may be 0*; for example, $(\boldsymbol{\nu}[])p$ and $\langle\rangle$ are valid process and list, respectively. We require that each linear term of a linear process appears* exactly once*.*

The informal meanings of the constructs are almost the same as that of the ordinary processes. The linear processes $\mathbf{0}$, $x_i\langle\lambda_1,\ldots,\lambda_n\rangle$ and $x_i(\mu_1,\ldots,\mu_n).p$ are nil process, output action and input prefixing, respectively. An important difference from the ordinary process is that, in linear processes, the output and input take lists of variables as arguments. When a list of linear names is received each element of a list must be used exactly once. There are two types of parallel composition $p \mid q$ and $p \parallel q$. The former is the conventional parallel composition and the latter is used when a replicated process is approximated by finite parallel compositions.[7] We use $\Pi_i p_i$ as a shorthand notation of $p_1 \parallel \cdots \parallel p_n$ and write the nullary composition of $\parallel$ as $\bot$. The approximation relation defined later (Section 4.2) may also help the readers to understand the intuitive meaning of linear processes.

We also identify processes with "similar structure". The *strong structural congruence*, written $p \equiv_0 q$ over linear processes is the smallest congruence relation that satisfies:

$$p \parallel q \equiv_0 q \parallel p \qquad (p \parallel q) \parallel r \equiv_0 p \parallel (q \parallel r)$$
$$(\boldsymbol{\nu}[\langle x_1, y_1\rangle,\ldots,\langle x_n, y_n\rangle])p \equiv_0 (\boldsymbol{\nu}[\langle x_{\sigma(1)}, y_{\sigma(1)}\rangle,\ldots,\langle x_{\sigma(n)}, y_{\sigma(n)}\rangle])p,$$

where σ is a permutation over $\{1,\ldots,n\}$. Given $I = \{i_1,\ldots,i_n\}$, we write $\boldsymbol{\nu}[\langle x_i, y_i\rangle]_{i\in I}$ for $\boldsymbol{\nu}[\langle x_{i_1}, y_{i_1}\rangle,\ldots,\langle x_{i_n}, y_{i_n}\rangle]$ because how the pairs $\langle x_{i_j}, y_{i_j}\rangle$ are ordered is inessential.

We now define the intersection types. The syntax of *raw types* and *raw (indexed) intersection types* are given by the following grammar:

$$\text{(Raw types)} \quad \rho ::= \mathbf{ch}^o_\alpha[\theta_1,\ldots,\theta_m] \mid \mathbf{ch}^i_\alpha[\theta_1,\ldots,\theta_n]$$
$$\text{(Raw intersection types)} \quad \theta ::= \bigwedge_{i\in I}(i,\rho_i)$$

where $I \subseteq_{\text{fin}} \mathbf{Nat}$ and α ranges over the set of *levels* $(\mathcal{A},\leq)$, a universal poset in which any finite poset can be embedded into. In the above grammar, an intersection $\bigwedge_{i\in I}(i,\rho_i)$ is a map $i \mapsto \rho_i$ from I to types. The intuitive meaning of $\bigwedge_{i\in I}(i,\rho_i)$ is the intersection $\rho_{i_1} \wedge \rho_{i_2} \wedge \cdots \wedge \rho_{i_n}$ provided that $I = \{i_1 < i_2 < \cdots < i_n\}$.

Levels express timing information, and types are defined as raw types with "appropriate levels". Let us write $\overline{\mathbf{lv}}(\rho)$ and $\overline{\mathbf{lv}}(\theta)$ for the set of levels that appear in ρ and θ, respectively. Then *types* and *intersection types* are inductively defined as follows: $\mathbf{ch}^m_\alpha[\theta_1,\ldots,\theta_n]$ $(m \in \{i,o\})$ is a type if θ_i is an intersection type for all $i \in \{1,\ldots,n\}$ and $\alpha \leq \gamma$ for all $\gamma \in \overline{\mathbf{lv}}(\theta_1,\ldots,\theta_n)$ and $\bigwedge_{i\in I}(i,\rho_i)$ is an intersection type if ρ_i is a type for all $i \in I$. *Hereafter, we use the metavariables ρ and θ to range over types and intersection types, respectively.*

Notations. We define $[n] \stackrel{\text{def}}{=} \{1,\ldots,n\}$ for a natural number n. A special symbol $\bullet$ is introduced to mean undefined type of sort T; now an intersection type θ can be also be represented by a (total) function from **Nat** to the union of the set of types and $\{\bullet\}$. We write $(i_1,\rho_{i_1}) \wedge \cdots \wedge (i_n,\rho_{i_n})$ for the intersection type θ such that $\mathrm{dom}(\theta) = \{i_1 < \cdots < i_n\}$

[7] (For readers familiar with resource calculi) Although the intuitive meaning of $p \parallel q$ is the parallel composition of p and q, this process should be thought of as an analogous to the bag in the resource λ-calculi [5, 10].

and $\theta(i_j) = \rho_{i_j}$ for every $j \in [n]$. We also write $\top$ for the empty intersection, i.e. θ such that $\theta(i) = \bullet$ for all $i \in \mathbf{Nat}$. The *dual* $\rho^\perp$ of type ρ is defined by $\mathbf{ch}^o_\alpha[\vec{\theta}]^\perp \stackrel{\text{def}}{=} \mathbf{ch}^i_\alpha[\vec{\theta}]$ and $\mathbf{ch}^i_\alpha[\vec{\theta}]^\perp_\alpha \stackrel{\text{def}}{=} \mathbf{ch}^o_\alpha[\vec{\theta}]$. We also define $\theta^\perp$ by $\theta^\perp(i) \stackrel{\text{def}}{=} (\theta(i))^\perp$. In what follows, we may often omit the annotations φ on the outputs because they are not needed as long as we are dealing with simple examples.

The type $\mathbf{ch}^i_\alpha[\theta_1, \dots, \theta_n]$ is for a channel that is used to receive n lists, where the i-th list has type θ_i and the type $\mathbf{ch}^o_\alpha[\vec{\theta}]$ is for output channels. If the i-th list has type $(i_1, \rho_1) \wedge \cdots \wedge (i_m, \rho_m)$, it means that the j-the element of the list has type ρ_j. For example, $a_i(\langle x_1, x_2\rangle, \langle \bar{y}_1\rangle).x_1().x_2().\bar{y}_1\langle\langle\rangle\rangle$ is well-typed if a_i has type $\mathbf{ch}^i_\alpha[(1, \mathbf{ch}^i_\beta[]) \wedge (2, \mathbf{ch}^i_\gamma[]), (1, \mathbf{ch}^o_\gamma[\top])]$ with $\alpha \le \beta \le \gamma$. As mentioned, the levels are used to describe the timing of actions. In the above example, the level γ tells us that the second element of the first argument, namely x_2, and the first element of the second argument, namely $\bar{y}_1$, must be used at the same timing. Levels also describe the fact that x_1 must be used before x_2 and $\bar{y}_1$ are used.

Although the intersection types are non-commutative in the sense that $(0, \rho) \wedge (1, \rho') \neq (0, \rho') \wedge (1, \rho)$, we consider that they are isomorphic. Intuitively, this means that we do not mind much about the order of elements in a list. For example, we consider that $\bar{a}_0\langle\langle x_0, x_1\rangle\rangle$ and $\bar{a}_0\langle\langle x_1, x_0\rangle\rangle$ are almost identical. Without this identification, we face a technical problem: an approximation of a forwarder $a_0(\langle y_0, y_1\rangle).\bar{b}_0\langle\langle y_1, y_0\rangle\rangle$ cannot be seen as an "identity" because $(\boldsymbol{\nu}[\langle\bar{a}_0, a_0\rangle])(a_0(\langle y_0, y_1\rangle).\bar{b}_0\langle\langle y_1, y_0\rangle\rangle \mid \bar{a}_0\langle\langle x_0, x_1\rangle\rangle)$ "reduces to" $\bar{b}_0\langle\langle x_1, x_0\rangle\rangle$. Another possible way to avoid this problem is to use fully *commutative* intersection types. We did not use this approach because, in a commutative type system, the relationship between linear processes and execution sequences becomes less precise.

▶ **Definition 8** (Type isomorphism). *We write $\varphi\colon \rho \cong \rho'$ (resp. $\varphi\colon \theta \cong \theta'$) to mean that ρ and ρ' (resp. θ and θ') are* isomorphic *and that φ is the* witness *of this isomorphism. This relation is defined by the rules below:*[8]

$$\frac{}{\mathrm{id}_\bullet\colon \bullet \cong \bullet}$$

$$\frac{\varphi_i\colon \theta_i \cong \theta'_i \qquad (\textit{for } i \in [n])}{\mathbf{ch}^o_\alpha[\varphi_1, \dots, \varphi_n]\colon \mathbf{ch}^o_\alpha[\theta'_1, \dots, \theta'_n] \cong \mathbf{ch}^o_\alpha[\theta_1, \dots, \theta_n]}$$

$$\frac{\varphi_i\colon \theta_i \cong \theta'_i \qquad (\textit{for } i \in [n])}{\mathbf{ch}^i_\alpha[\varphi_1, \dots, \varphi_n]\colon \mathbf{ch}^i_\alpha[\theta_1, \dots, \theta_n] \cong \mathbf{ch}^i_\alpha[\theta'_1, \dots, \theta'_n]}$$

$$\frac{\sigma\colon \mathbf{Nat} \xrightarrow{\cong} \mathbf{Nat} \qquad \varphi_i\colon \rho_i \cong \rho'_{\sigma(i)} \qquad (\textit{for } i \in \mathbf{Nat})}{(\sigma, (\varphi_i)_{i\in\mathbf{Nat}})\colon \bigwedge_{i\in\mathbf{Nat}} (i, \rho_i) \cong \bigwedge_{i\in\mathbf{Nat}} (i, \rho'_i)}$$

▶ Remark 9. The reason for annotating arguments of free outputs with φ is quite technical. The notion of type isomorphism was taken from the rigid intersection type system given by Tsukada et al. [21], but in their calculus, witnesses do not appear in the syntax. This is so because all the (raw) terms in their resource calculus are assumed to be in η-long form. (See [21] for details.)

Similarly, we may remove witnesses of type isomorphisms from our linear calculus if there is a way to convert a linear process p to an "equivalent" process p' that does not contain any free outputs. A possible way to do this is to transform a free output to a "bound

[8] Here, $\bigwedge_{i\in I}(i, \rho_i)$ is considered as a total map $\bigwedge_{i\in\mathbf{Nat}}(i, \rho_i)$ in which $\rho_i \stackrel{\text{def}}{=} \bullet$ if $i \notin I$.

$$\frac{\varphi_i : \theta_i \cong \theta'_i \qquad \varphi_i \cdot x^i = \lambda_i \quad (\text{for } i \in [n]) \quad \alpha \le \overline{\mathbf{lv}}(\theta_1, \dots, \theta_n)}{x^1 : \theta_1 \sqcap \dots \sqcap x^n : \theta_n, \bar{a} : (i, \mathbf{ch}^o_\alpha[\theta'_1, \dots, \theta'_n]) \vdash_\alpha \bar{a}_i\langle \lambda_1, \dots, \lambda_n \rangle} \ (\textsc{TOut})$$

$$\frac{\Gamma, x^1 : \theta_1, \dots, x^n : \theta_n \vdash_\beta p \quad \mathrm{id}_{\theta_i} \cdot x^i = \mu_i \quad \alpha \le \beta}{\Gamma \sqcap a : (i, \mathbf{ch}^i_\beta[\theta_1, \dots, \theta_n]) \vdash_\alpha a_i(\mu_1, \dots, \mu_n).p} \ (\textsc{TIn}) \qquad \frac{\Gamma \vdash_\beta p \quad \alpha \le \beta}{\Gamma \sqcap \tau : (i, \mathbf{ch}^i_\beta[]) \vdash_\alpha \tau_i.p} \ (\textsc{TTau})$$

$$\frac{\Gamma_1 \vdash_\alpha p_1 \qquad \Gamma_2 \vdash_\alpha p_2}{\Gamma_1 \sqcap \Gamma_2 \vdash_\alpha p_1 \mid p_2} \ (\textsc{TPar}) \qquad \frac{\Gamma_i \vdash_\alpha p_i \quad (1 \le i \le n)}{\Gamma_1 \sqcap \dots \sqcap \Gamma_n \vdash_\alpha p_1 \parallel \dots \parallel p_n} \ (\textsc{TRep})$$

$$\frac{}{\emptyset \vdash_\alpha \mathbf{0}} \ (\textsc{TNil}) \qquad \frac{\Gamma, \bar{a} : \theta, a : \theta^\perp \vdash_\alpha p}{\Gamma \vdash_\alpha (\boldsymbol{\nu}[\langle \bar{a}_i, a_i \rangle]_{i \in \mathrm{dom}(\theta)})p} \ (\textsc{TNu})$$

Figure 2 Typing rules for the intersection type system.

output + forwarder" (cf. [4]). That is to (recursively) transform a free output $\bar{a}_0\langle\langle \bar{b}_0 \rangle\rangle$ into $(\boldsymbol{\nu}[\langle \bar{c}_0, c_0 \rangle])(\bar{a}_0\langle\langle \bar{c}_0 \rangle\rangle \mid b(\mu).\bar{c}_0\langle\mu\rangle)$. We chose to keep free outputs in the syntax of the linear process because by doing so, it is easier to see the correspondence between a (non-linear) process P, which may contain free outputs, and its linear approximation p. Note that we cannot assume that (non-linear) processes do not contain free outputs because the validity of the transformation $\bar{a}\langle b \rangle = (\boldsymbol{\nu}\bar{c}c)(\bar{a}\langle \bar{c} \rangle \mid c \hookrightarrow \bar{b})$ is not something that is taken for granted (even if the forwarder does not introduce any delay). Actually, the above translation is an instance of the rule (2) that we aim to validate in this work.

We define yet another operator $\theta_1 \sqcap \theta_2$ for intersection types which "coalesces" the two intersection. It is defined by $(\theta_1 \sqcap \theta_2)(i) \stackrel{\text{def}}{=} \theta_1(i)$ if $i \in \mathrm{dom}(\theta_1)$, $(\theta_1 \sqcap \theta_2)(i) \stackrel{\text{def}}{=} \theta_2(i)$ if $i \in \mathrm{dom}(\theta_2)$ and $(\theta_1 \sqcap \theta_2)(i) \stackrel{\text{def}}{=} \bullet$ otherwise, provided that $\mathrm{dom}(\theta_1) \cap \mathrm{dom}(\theta_2) = \emptyset$.

A *type environment*, often denoted by Γ, is a finite set of pairs of the form $t : \theta$ with $\theta \neq \top$ such that $(t^1 : \theta_1), (t^2 : \theta_2) \in \Gamma$ implies $t^1 \neq t^2$. For notational convenience, we may write $\Gamma, x : \top$ to express Γ, i.e. allow $\top$ to appear in a type environment. We define $\mathrm{dom}(\Gamma)$ as $\{t \mid \exists\theta.\ (t : \theta) \in \Gamma\}$ and $\Gamma(t)$ by $\Gamma(t) \stackrel{\text{def}}{=} \theta$ if $(t : \theta) \in \Gamma$ and $\Gamma(t) \stackrel{\text{def}}{=} \top$ otherwise. For type environments Γ_1 and Γ_2, $\Gamma_1 \sqcap \Gamma_2$ is defined by pointwise extension of $\sqcap$, that is $\Gamma_1 \sqcap \Gamma_2 \stackrel{\text{def}}{=} \{(t : \theta) \mid t \in \mathrm{dom}(\Gamma_1) \cup \mathrm{dom}(\Gamma_2), \theta = \Gamma_1(t) \sqcap \Gamma_2(t)\}$ provided that $\Gamma_1(t) \sqcap \Gamma_2(t)$ is defined for $t \in \mathrm{dom}(\Gamma_1) \cup \mathrm{dom}(\Gamma_2)$; otherwise $\Gamma_1 \sqcap \Gamma_2$ is undefined.

We consider judgments of the form $\Gamma \vdash_\alpha p$ and the typing rules are given in Figure 2. We stipulate that the deduction is allowed only if the result of the $\sqcap$ operation in the conclusion is defined. The operation $\varphi \cdot x$ used in the above definition is defined by

$$(\sigma, (\varphi_i)_{i \in \mathbf{Nat}}) \cdot x \stackrel{\text{def}}{=} \langle \varphi_{\sigma^{-1}(i_1)} \cdot x_{\sigma^{-1}(i_1)}, \dots, \varphi_{\sigma^{-1}(i_n)} \cdot x_{\sigma^{-1}(i_n)} \rangle,$$

where $(\sigma, (\varphi_i)_{i \in \mathbf{Nat}}) : \theta \cong \theta'$ and $\mathrm{dom}(\theta') = \{i_1 < \dots < i_n\}$; similarly $\mathrm{id}_\theta \cdot x$ is also used to express $\langle x_{i_1}, \dots, x_{i_n} \rangle$ when $\mathrm{dom}(\theta) = \{i_1 < \dots < i_n\}$.

Let us explain how the subscript α of $\vdash_\alpha$ is used; the other parts of the typing rule should be easy to understand. The intuitive meaning of the subscript α of $\vdash_\alpha$ is the "current time". The typing rule for output actions ensures that the "level of $\bar{a}_i$" is the "current time", that is the rule ensures that the output cannot be delayed. On the other hand, we may delay an input or a τ action. For example, in the rule (TIn), the "level of a_i" can be greater than α meaning that we can delay the use of a_i. The rule (TIn) also says that the "level of a_i" must be equal to the level assigned to $\Gamma, x^1 : \theta_1, \dots, x^n : \theta_n \vdash_\beta p$. This expresses the fact that the unguarded outputs in p must be used as soon as a_i is used, i.e. there cannot be any delay between an input and an output.

$$\frac{}{x : \{i_1, \ldots, i_n\} \vdash \langle \varphi_1 \cdot x_{i_1}, \ldots, \varphi_n \cdot x_{i_n} \rangle \sqsubset x} \qquad \frac{}{\vdash \mathbf{0} \sqsubset \mathbf{0}} \qquad \frac{}{\vdash \bot \sqsubset \tau.P}$$

$$\frac{X_j \vdash \lambda_j \sqsubset x^j \quad (\text{for } j \in [n])}{X_1 \sqcap \cdots \sqcap X_n \sqcap \bar{a} : \{i\} \vdash \bar{a}_i\langle \lambda_1, \ldots, \lambda_n \rangle \sqsubset \bar{a}\langle x^1, \ldots, x^n \rangle} \qquad \frac{X \vdash p \sqsubset P}{X \sqcap \tau : \{i\} \vdash \tau_i.p \sqsubset \tau.P}$$

$$\frac{X, x^1 : S_1, \ldots, x^n : S_n \vdash p \sqsubset P \qquad x^j : S_j \vdash \mu_j \sqsubset x^j \quad (\text{for } j \in [n])}{X \sqcap a : \{i\} \vdash a_i(\mu_1, \ldots, \mu_n).p \sqsubset a(x^1, \ldots, x^n).P}$$

$$\frac{X_1 \vdash p \sqsubset P \qquad X_1 \vdash q \sqsubset Q}{X_1 \sqcap X_2 \vdash p \mid q \sqsubset P \mid Q} \qquad \frac{X_i \vdash p_i \sqsubset P \quad (\text{for } i \in [n])}{X_1 \sqcap \cdots \sqcap X_n \vdash p_1 \parallel \cdots \parallel p_n \sqsubset !P}$$

$$\frac{X, x : S, y : S \vdash p \sqsubset P \qquad S = \{i_1, \ldots, i_n\} \qquad \rho_i \sqsubset T \quad (\text{for } i \in S)}{X \vdash (\boldsymbol{\nu}[\langle x_{i_1}, y_{i_1} \rangle_{\rho_{i_1}}, \ldots, \langle x_{i_n}, y_{i_n} \rangle_{\rho_{i_n}}])p \sqsubset (\boldsymbol{\nu}_T xy)P}$$

Figure 3 Rules for approximation relation. We stipulate that the deduction is allowed only if the result of the $\sqcap$ operation in the conclusion is defined.

4.2 Approximation

In this subsection we show how sorts are refined by intersection types and processes are approximated by linear processes.

Given a sort T, the *refinement relation* $\rho \sqsubset T$ (resp. $\theta \sqsubset T$), meaning that the type ρ (resp. the intersection type θ) refines the sort T, is defined by the following rules:

$$\frac{\theta_i \sqsubset T_i \quad (i \in [n]) \qquad m \in \{i, o\}}{\mathbf{ch}^m_\alpha[\theta_1, \ldots, \theta_n] \sqsubset \mathbf{ch}^m[T_1, \ldots, T_n]} \qquad \frac{\rho_i \sqsubset T \quad (i \in I \subseteq_{\text{fin}} \mathbf{Nat})}{\bigwedge_{i \in I}(i, \rho_i) \sqsubset T}.$$

We write $\Gamma \sqsubset \Delta$ if $(x : \theta) \in \Gamma$ implies that $(x : T) \in \Delta$ for some T and $\theta \sqsubset T$.

Next we show how processes are approximated by linear processes.

A *term refinement* X is a finite set of the form $t^1 : S_1, \ldots, t^n : S_n$ such that $S_i \subseteq_{\text{fin}} \mathbf{Nat}$ and $i \neq j$ implies $t^i \neq t^j$, where each t^i is a (non-linear) channel name or the constant τ. The set S_i expresses how many times t^i is used in the approximation. Notations $X(t)$ and $X_1 \sqcap X_2$ are defined analogous to $\Gamma(t)$ and $\Gamma_1 \sqcap \Gamma_2$. There is a canonical way to obtain a term refinement from a type environment: given a type environment Γ, we define $\Gamma^\natural$ as $\{(t : \mathrm{dom}(\Gamma(t))) \mid t \in \mathrm{dom}(\Gamma)\}$.

An *approximation judgement* is of the form $X \vdash p \sqsubset P$ and inference rules for judgments are given in Figure 3. It should be emphasized that we do not allow $\bot \sqsubset \bar{a}\langle \vec{x} \rangle$, that is we ensure that all the output actions are used. Note that we can discard an output action that is guarded by τ, i.e. $\bot \sqsubset \tau.\bar{a}\langle \vec{x} \rangle$, and this is why the translation $(-)^\dagger$ defined in Section 2 allows us to relate the reduction $\longrightarrow$ with $\Rightarrow$.

4.3 Reduction

This subsection defines the reduction relation for linear processes. We also show that every reduction sequence from P has a representation by a linear process that approximates P.

The reduction relation of linear processes is almost the same as that of processes except for the fact that we take actions of type isomorphisms to linear processes into account. The action of φ to linear processes is defined by the rules in Figure 4. It is defined via the action of type isomorphisms on subject names and operation $\{\varphi \cdot y/x\}$, which substitutes

$$\langle \varphi_1 \cdot x_{i_1}, \dots, \varphi_k \cdot (\varphi' \cdot x_{i_k}), \dots, \varphi_n \cdot x_{i_n} \rangle \stackrel{\text{def}}{=} \langle \varphi_1 \cdot x_{i_1}, \dots, (\varphi_k \circ \varphi') \cdot x_{i_k}, \dots, \varphi_n \cdot x_{i_n} \rangle$$
$$(\mathbf{ch}^o[\varphi] \cdot \bar{a}_i)\langle\langle \varphi'_{i_1} \cdot x_{i_1}, \dots, \varphi'_{i_n} \cdot x_{i_n} \rangle\rangle \stackrel{\text{def}}{=} \bar{a}_i\langle\langle (\varphi_{i_1} \circ \varphi'_{\sigma(i_1)}) \cdot x_{\sigma(i_1)}, \dots, (\varphi_{i_n} \circ \varphi'_{\sigma(i_n)}) \cdot x_{\sigma(i_n)} \rangle\rangle$$
$$(\mathbf{ch}^i[\varphi] \cdot a_i)(\langle x_{i_1}, \dots, x_{i_n} \rangle).p \stackrel{\text{def}}{=} a_i(\langle x_{i_1}, \dots, x_{i_n} \rangle).p\{\varphi_{\sigma^{-1}(j)} \cdot x_{\sigma^{-1}(j)}/x_j\}_{j \in \{i_1, \dots, i_n\}}$$

Figure 4 Action of isomorphisms on linear (monadic) processes where $\varphi = (\sigma, (\varphi_i))$ in the last two equations; the action on polyadic processes is defined similarly.

$\varphi \cdot y$ to x. The substitution $\{\varphi \cdot y/x\}$ works as the standard substitution, except for the fact the action of φ is performed after the substitution. The witness $\varphi_2 \circ \varphi_1 \colon \rho_1 \cong \rho_3$ is the *composition* of $\varphi_1 \colon \rho_1 \cong \rho_2$ and $\varphi_2 \colon \rho_2 \cong \rho_3$, which is defined as in the case of rigid resource calculus [21]. (The definition of $\varphi_2 \circ \varphi_1$ is not necessary to understand the following content; see Appendix B.1 for the definition.) For readability, given $\lambda \stackrel{\text{def}}{=} \langle \varphi_1 \cdot y_1, \dots, \varphi_n \cdot y_n \rangle$ and $\mu \stackrel{\text{def}}{=} \langle x_1, \dots, x_n \rangle$, we write $\{\lambda/\mu\}$ to denote $\{\varphi_1 \cdot y_1/x_1, \dots, \varphi_n \cdot y_n/x_n\}$.

The *structural precongruence* $\Rrightarrow$ over linear process is the smallest *precongruence* relation that contains $\equiv_0$, contains α-equivalence and satisfies:

$$\mathbf{0} \mid p \Longleftrightarrow p \qquad p \mid q \Longleftrightarrow q \mid p \qquad (p \mid q) \mid r \Longleftrightarrow p \mid (q \mid r)$$
$$(\boldsymbol{\nu}[\langle \vec{w}, \vec{z} \rangle])(\boldsymbol{\nu}[\langle \vec{y}, \vec{z} \rangle])p \Longleftrightarrow (\boldsymbol{\nu}[\langle \vec{y}, \vec{z} \rangle])(\boldsymbol{\nu}[\langle \vec{w}, \vec{x} \rangle])p \qquad (\mathbf{fn}(\vec{w}, \vec{x}) \cap \mathbf{fn}(\vec{y}, \vec{z}) = \emptyset)$$
$$(\boldsymbol{\nu}[\langle x_1, y_1 \rangle, \dots, \langle x_n, y_n \rangle])p \mid q \Rrightarrow (\boldsymbol{\nu}[\langle x_1, y_1 \rangle, \dots, \langle x_n, y_n \rangle])(p \mid q) \quad (\vec{x}, \vec{y} \notin \mathbf{fn}(q))$$

where $p \Longleftrightarrow q$ means $p \Rrightarrow q$ and $q \Rrightarrow p$. The *structural congruence* $\equiv$ for linear processes is defined as symmetric closure of $\Rrightarrow$.

We define the *one-step reduction relation* over well-typed linear processes by the base rule

$$(\boldsymbol{\nu}\vec{\xi})(\boldsymbol{\nu}[\langle \bar{a}_j, a_j \rangle]_{j \in J})(\Pi_{i \in I} a_i(\mu_{i1}, \dots, \mu_{in_i}).p_i \mid \bar{a}_m\langle \lambda_1, \dots, \lambda_n \rangle \mid q) \xrightarrow{0}$$
$$(\boldsymbol{\nu}\vec{\xi})(\boldsymbol{\nu}[\langle \bar{a}_j, a_j \rangle]_{j \in J'})(\Pi_{i \in I'} a_i(\mu_{i1}, \dots, \mu_{in_i}).p_i \mid p_m\{\lambda_1/\mu_{m1}, \dots, \lambda_n/\mu_{mn}\} \mid q)$$

where $(\boldsymbol{\nu}\vec{\xi})$ is a sequence of name restrictions, $m \in I \subseteq J$, $J' = J \setminus \{m\}$ and $I' = I \setminus \{m\}$, and the structural rule which concludes $p \xrightarrow{0} q$ from $p \Rrightarrow p'$ and $p' \xrightarrow{0} q$. The relation $\xrightarrow{\tau}$ is obtained by replacing the base rule of the $\xrightarrow{0}$ with $(\boldsymbol{\nu}\vec{\xi})(\tau_i.p \mid q) \xrightarrow{\tau} (\boldsymbol{\nu}\vec{\xi})(p \mid q)$.

▶ Remark 10. We use $\Rrightarrow$ instead of $\equiv$ in the definition of reduction because $X \vdash p \sqsubset P$ and $p \equiv q$ does not ensure the existence of Q such that $X \vdash q \sqsubset Q$ and $P \equiv Q$. For instance, if $P \stackrel{\text{def}}{=} (\boldsymbol{\nu}\bar{a}a)(!a(x).R \mid \tau.\bar{a}\langle y \rangle)$ then $(\boldsymbol{\nu}[])(\bot \mid \bot)$ approximates P and this linear process is structurally congruent to $(\boldsymbol{\nu}[])\bot \mid \bot$, but there is no Q such that $(\boldsymbol{\nu}[])\bot \mid \bot \sqsubset Q$ and $P \equiv Q$.

We now show the relationship between execution sequences and linear approximations. Let us write $P \xrightarrow{\pi} Q$ if there exists a sequence $P = P_0 \xrightarrow{l_1} P_1 \xrightarrow{l_2} \cdots \xrightarrow{l_n} P_n = Q$, where each l_i is either 0 or τ, and $\pi = l_1 l_2 \dots l_n$; $p \xrightarrow{\pi} q$ is defined similarly. We write $(p \xrightarrow{\pi} q) \sqsubset (P \xrightarrow{\pi} Q)$ if there exists $p = p_0 \xrightarrow{l_1} \cdots \xrightarrow{l_n} p_n = q$ and $P = P_0 \xrightarrow{l_1} \cdots \xrightarrow{l_n} P_n = Q$ such that $X_i \vdash p_i \sqsubset P_i$ for some X_i for each $i \in \{0, \dots, n\}$ and $\pi = l_1 \cdots l_n$.

▶ **Proposition 11.** *Let $\tau : \mathbf{ch}^i[] \vdash P$, i.e. let P be a process without any free names.*

1. *Suppose that $\Gamma \vdash_\alpha p$ and $\Gamma^\natural \vdash p \sqsubset P$. If $p \xrightarrow{\pi} q$ then we have $(p \xrightarrow{\pi} q) \sqsubset (P \xrightarrow{\pi} Q)$ for some Q.*
2. *Assume $P \xrightarrow{\pi} Q$, $\Gamma \vdash_\alpha q$ and $\Gamma^\natural \vdash q \sqsubset Q$. Then we have $(p \xrightarrow{\pi} q) \sqsubset (P \xrightarrow{\pi} Q)$ for some p.*

5 LTS based on linear approximations

Using the notion of linear processes, we introduce a labelled transition system (LTS) for processes in the form of a presheaf to describe the behaviour of processes in which outputs cannot be delayed. Intuitively, the LTS that describes the behaviour of P is given as an LTS whose states are linear approximations of P and transition relation is the extension relation $\trianglelefteq$, which we briefly explained in Section 3. This LTS will be presented as a presheaf following the view that presheaves can be regarded as transition systems [9, 23].

5.1 Extension relation

We now define an ordering $p' \trianglelefteq p$ over linear processes, which may be read as "p extends p'". Giving a larger linear approximation corresponds to extending an execution sequence.

Before we define the extension relation on linear processes, we define the extension relation over types.

▶ **Definition 12.** *Let A be a set of levels.* Restriction of types and intersection types *are inductively defined by:*

$$\mathbf{ch}^o_\alpha[\theta_1,\dots,\theta_n]\restriction_A \stackrel{\text{def}}{=} \begin{cases} \mathbf{ch}^o_\alpha[\theta_1\restriction_A,\dots,\theta_n\restriction_A] & (\textit{if } \alpha \in A) \\ \bullet & (\textit{otherwise}) \end{cases} \quad \textit{and } (\theta\restriction_A)(i) \stackrel{\text{def}}{=} \theta(i)\restriction_A.$$

where restrictions over input types are defined similar to that of output types. The restriction of type isomorphisms $\varphi\restriction_A$ is defined in a similar manner. (See the appendix for details.) We write $\rho' <: \rho$ and if $\rho' = \rho\restriction_A$ for some $A \subseteq \mathcal{A}$ and $\varphi' <: \varphi$ if $\varphi' = \varphi\restriction_A$ for some $A \subseteq \mathcal{A}$.

The *extension relation* on linear processes, written $p' \trianglelefteq p$, is inductively defined by the rules in Figure 5. For example, $a_1(\langle\rangle).\bot \trianglelefteq a_1(\langle x_1\rangle).\tau_1.x_1\langle\rangle$ holds and this intuitively means that $!a(x).x\langle\rangle \xrightarrow{a(x)} !a(x).x\langle\rangle \mid x\langle\rangle$ can be extended to $!a(x).x\langle\rangle \xrightarrow{a(x)} !a(x).x\langle\rangle \mid x\langle\rangle \xrightarrow{x\langle\rangle} !a(x).x\langle\rangle \mid \mathbf{0}$ (under the assumption that both of the linear processes approximate $!a(x).x\langle\rangle$).

Extending a linear process does not precisely correspond to extending an execution sequence: there are cases where an execution sequence cannot be extended even if the corresponding linear process can be extended. This problem is due to the existence of deadlocks. For instance, we have $(\boldsymbol{\nu}[])(\boldsymbol{\nu}[])(\bot \mid \bot) \trianglelefteq (\boldsymbol{\nu}[\langle\bar{a}_1, a_1\rangle])(\boldsymbol{\nu}[\langle\bar{b}_1, b_1\rangle])(a_1.\tau_1.\bar{b}_1 \mid b_1.\tau_2.\bar{a}_1)$, but both of the linear processes are not reducible. To exclude linear processes that may create a deadlock, we introduce the notion of terminable processes:

▶ **Definition 13.** *A linear process p is* idle *if it has no action (input, output nor τ), i.e. consisting of 0, $\bot$, $\mid$ and $\boldsymbol{\nu}[]$. A linear process p is* terminable *if $(\boldsymbol{\nu}\vec{\xi}).(p \mid q) \xrightarrow{0}{}^{*} r$ for some $\vec{\xi}$, q and idle r.*

Only terminable processes will be used as the states of the LTS.

In case p and p' correspond to executions that only consists of $\xrightarrow{0}$ and $\xrightarrow{\tau}$ the intuition that $p \trianglelefteq p'$ corresponds to "extending execution sequences" can be formalised as follows:

▶ **Proposition 14.** *Let $\tau : \mathbf{ch}^i[] \vdash P$ and let $\mathcal{R}$ be a relation between execution sequences starting from P and well-typed terminable linear approximations of P such that $(P \xrightarrow{\pi} Q)\ \mathcal{R}\ p$ if and only if $(p \xrightarrow{\pi} q) \sqsubset (P \xrightarrow{\pi} Q)$ for a process q that is typed under the empty environment. Then if $(P \xrightarrow{\pi} Q)\ \mathcal{R}\ p$*

1. *$Q \xrightarrow{\pi'} Q'$ implies that $(P \xrightarrow{\pi} Q \xrightarrow{\pi'} Q')\ \mathcal{R}\ p'$ and $p \trianglelefteq p'$ for some p'.*
2. *if $p \trianglelefteq p'$ for some terminable well-typed linear process p' that approximates P, then there is an execution $Q \xrightarrow{\pi'} Q'$ such that $(P \xrightarrow{\pi} Q \xrightarrow{\pi'} Q')\ \mathcal{R}\ p'$.*

$$\frac{I = \{i_1 < \cdots < i_m\} \qquad J = \{j_1 < \cdots < j_n\} \qquad J \subseteq I \qquad \varphi'_i <: \varphi_i \quad (\text{for } i \in J)}{\langle \varphi'_{j_1} \cdot x_{j_1}, \ldots, \varphi'_{j_n} \cdot x_{j_n} \rangle \trianglelefteq \langle \varphi_{i_1} \cdot x_{i_1}, \ldots, \varphi_{i_m} \cdot x_{i_m} \rangle}$$

$$\frac{}{\mathbf{0} \trianglelefteq \mathbf{0}} \qquad \frac{}{\bot \trianglelefteq \tau_i.p} \qquad \frac{p \trianglelefteq q}{\tau_i.p \trianglelefteq \tau_i.q} \qquad \frac{J \subseteq I \qquad \rho'_i <: \rho_i \quad (\text{for } i \in J) \qquad p \trianglelefteq q}{(\boldsymbol{\nu}[\langle \bar{a}_i, a_i \rangle_{\rho'_i}]_{i \in J})q \trianglelefteq (\boldsymbol{\nu}[\langle \bar{a}_i, a_i \rangle_{\rho_i}]_{i \in I})p}$$

$$\frac{\lambda'_j \trianglelefteq \lambda_j \quad (\text{for } j \in [n])}{\bar{a}_i\langle \lambda'_1, \ldots, \lambda'_n \rangle \trianglelefteq \bar{a}_i\langle \lambda_1, \ldots, \lambda_n \rangle} \qquad \frac{p \trianglelefteq q \qquad \mu'_j \trianglelefteq \mu_j \quad (\text{for } j \in [n])}{a_i(\mu'_1, \ldots, \mu'_n).p \trianglelefteq a_i(\mu_1, \ldots, \mu_n).q}$$

$$\frac{}{\bot \trianglelefteq a_i(\vec{\mu}).q} \qquad \frac{p' \trianglelefteq p \qquad q' \trianglelefteq q}{p' \mid q' \trianglelefteq p \mid q} \qquad \frac{m \leq n \qquad p'_i \trianglelefteq p_i \quad p'_i \neq \bot \quad (\text{for } i \in [m])}{p'_1 \parallel \cdots \parallel p'_m \trianglelefteq p_1 \parallel \cdots \parallel p_n}$$

Figure 5 Rules for extension relation. Here we identify processes up to $\equiv_0$.

5.2 Presheaf semantics

We define the LTS of $\Delta \vdash P$ as a presheaf $[\![P]\!] \colon \mathcal{E}_\Delta \to \mathbf{Sets}$. Roughly speaking, the path category $\mathcal{E}_\Delta$ is a category of type environments that refines Δ and $[\![P]\!]$ maps a type environment Γ to the set of approximations of P that is typed under Γ.

Actually, the objects of the path category are not only type environments, but the pair of type environments and the "current time".

▶ **Definition 15.** *We say that* (Γ, α) extends (Γ', α) *and write* $(\Gamma', \alpha) <: (\Gamma, \alpha)$ *if there exists a* witness $A \subseteq \overline{\mathbf{lv}}(\Gamma) \cup \{\alpha\}$ *that satisfies (i)* $\mathrm{dom}(\Gamma') \subseteq \mathrm{dom}(\Gamma)$ *and* $\Gamma'(t) = \Gamma(t)\restriction_A$*, for* $t \in \mathrm{dom}(\Gamma)$*, (ii)* $\alpha \in A$ *and (iii)* A *is downward-closed: for every* β, γ *appearing in* Γ*,* $\beta \leq \gamma$ *and* $\gamma \in A$ *implies* $\beta \in A$*.*
We define the *category of type environments* $\mathcal{E}_\Delta$ to be a category whose objects are (Γ, α) such that $\Gamma \sqsubset \Delta$ and whose morphisms are given by the relation $(\Gamma', \alpha) <: (\Gamma, \alpha)$.

We now define the presheaf $[\![P]\!]$. Given $\Delta \vdash P$ and $\Gamma \sqsubset \Delta$, the set $[\![P]\!](\Gamma, \alpha)$ is defined by $[\![P]\!](\Gamma, \alpha) \stackrel{\text{def}}{=} \{p \mid \Gamma^\natural \vdash p \sqsubset P, \Gamma \vdash_\alpha p \text{ and } p \text{ is terminable}\}$. (Here we are identifying linear processes up to $\equiv_0$.)

▶ **Proposition 16.** *Assume that* $\Gamma \vdash_\alpha p$*,* p *is terminable and* $(\Gamma', \alpha) <: (\Gamma, \alpha)$*. Then there is a unique (up to* $\equiv_0$*) linear process that satisfy* $q \trianglelefteq p$ *and* $\Gamma' \vdash_\alpha q$*.*

By Proposition 16 there is a map $[\![P]\!](-,-)$ that maps an extension relation $(\Gamma', \alpha) <: (\Gamma, \alpha)$ to a function from $[\![P]\!](\Gamma, \alpha)$ to $[\![P]\!](\Gamma', \alpha)$ that maps $p \in [\![P]\!](\Gamma, \alpha)$ to q such that $q \trianglelefteq p$ and $\Gamma' \vdash_\alpha q$. Given $\Gamma \vdash_\alpha p$, we will write $p\restriction_{\Gamma', \alpha}$ for the process that is uniquely determined by the above proposition, provided that $(\Gamma', \alpha) <: (\Gamma, \alpha)$.

▶ **Theorem 17.** *Let* $\Delta \vdash P$*. Then* $[\![P]\!](-,-)$ *is a functor from* $\mathcal{E}_\Delta$ *to* **Sets**.

▶ **Example 18.** Consider a process $P \stackrel{\text{def}}{=} (\boldsymbol{\nu} \bar{a}a)(!a(x).\tau.\bar{z}\langle\rangle \mid !a(x).\tau.x\langle \bar{y} \rangle \mid \bar{a}\langle \bar{w} \rangle)$ such that $\Delta \vdash P$, where $\Delta \stackrel{\text{def}}{=} \tau : \mathbf{ch}^i[], \bar{w} : \mathbf{ch}^o[\mathbf{ch}^o[]], \bar{y} : \mathbf{ch}^o[], \bar{z} : \mathbf{ch}^o[]$. Then we have

$$[\![P]\!](\Gamma_1, \alpha) = \{(\boldsymbol{\nu}[\langle \bar{a}_1, a_1 \rangle])(a_1(\langle\rangle).\bot \mid \bot \mid \bar{a}_1\langle\langle\rangle\rangle), (\boldsymbol{\nu}[\langle \bar{a}_1, a_1 \rangle])(\bot \mid a_1(\langle\rangle).\bot \mid \bar{a}_1\langle\langle\rangle\rangle)\}$$
$$[\![P]\!](\Gamma_2, \alpha) = \{(\boldsymbol{\nu}[\langle \bar{a}_1, a_1 \rangle])(\bot \mid a_1(\langle x_1 \rangle).\tau_1.x_1\langle\langle\rangle\rangle \mid \bar{a}_1\langle\langle \bar{w}_1 \rangle\rangle)\}$$
$$[\![P]\!](\Gamma_3, \alpha) = \{(\boldsymbol{\nu}[\langle \bar{a}_1, a_1 \rangle])(\bot \mid a_1(\langle x_1 \rangle).\tau_1.x_1\langle\langle \bar{y}_1 \rangle\rangle \mid \bar{a}_1\langle\langle \bar{w}_1 \rangle\rangle)\}$$

for $\Gamma_1 \stackrel{\text{def}}{=} \emptyset$, $\Gamma_2 \stackrel{\text{def}}{=} \tau\colon (1, \mathbf{ch}^i_\beta[]), \bar{w} : (1, \mathbf{ch}^o_\beta[\top])$ and $\Gamma_3 \stackrel{\text{def}}{=} \tau\colon (1, \mathbf{ch}^i_\beta[]), \bar{w} : (1, \mathbf{ch}^o_\beta[(1, \mathbf{ch}^o_\gamma[])]), \bar{y} : (1, \mathbf{ch}^o_\gamma[])$ with $\alpha < \beta < \gamma$. Note that $(\Gamma_2, \alpha) <: (\Gamma_3, \alpha)$ because we can take $\{\alpha, \beta\}$ as the witness. We also have $(\Gamma_1, \alpha) <: (\Gamma_2, \alpha)$ since $\{\alpha\}$ is a witness. The function $[\![P]\!]((\Gamma_1, \alpha) <: (\Gamma_2, \alpha))$ maps the only linear process of $[\![P]\!](\Gamma_2, \alpha)$ to the linear process $(\boldsymbol{\nu}[\langle a_1, \bar{a}_1\rangle])(\bot \mid a_1(\langle\rangle).\bot \mid \bar{a}\langle\langle\rangle\rangle)$.

6 $\simeq^c_\tau$ is a π_F-theory

As explained in Section 3, to prove Theorem 4, it suffices to show that (i) $\simeq^c_\tau$ satisfies the axioms such as (2) and (3), and (ii) that barbed congruence is a congruence relation, which trivially holds. Instead of directly proving (i), we define a yet another equivalence $\sim$ and show that $\sim$ is a congruence relation that satisfies the axioms and $\sim \subseteq \simeq^c_\tau$. These are relatively easier to show than to directly prove (i).

The equivalence $\sim$ is defined using the notion of *open map bisimulation* [12].[9] We write $P \sim Q$ if and only if $[\![P]\!]$ and $[\![Q]\!]$ are open map bisimilar, i.e. if there is a span $[\![P]\!] \xleftarrow{f} X \xrightarrow{g} [\![Q]\!]$, where f and g are open maps. A map $f\colon [\![P]\!] \to [\![Q]\!]$ is called an *open map* if for every $m\colon y(\Gamma_1, \alpha_1) \to y(\Gamma, \alpha_2)$, making the square below commute

$$\begin{array}{ccc} y(\Gamma_1,\alpha) & \xrightarrow{p} & [\![P]\!] \\ \downarrow{\scriptstyle m} & & \downarrow{\scriptstyle f} \\ y(\Gamma_2,\alpha) & \xrightarrow{q} & [\![Q]\!] \end{array} \qquad \text{there is a diagonal map } d \qquad \begin{array}{ccc} y(\Gamma_1,\alpha) & \xrightarrow{p} & [\![P]\!] \\ \downarrow{\scriptstyle m} & \overset{d}{\nearrow} & \downarrow{\scriptstyle f} \\ y(\Gamma_2,\alpha) & \xrightarrow{q} & [\![Q]\!] \end{array}$$

making the two triangles commute.

Showing that there is an open map $f\colon [\![P]\!] \to [\![Q]\!]$ is analogous to giving a functional bisimulation (indexed by (Γ, α)) between $[\![P]\!](\Gamma, \alpha)$ and $[\![Q]\!](\Gamma, \alpha)$. The naturality of f means that f is a simulation because the naturality says $f(p)\restriction_{\Gamma_1,\alpha} = f(p\restriction_{\Gamma_1,\alpha})$. The morphism f being open ensures that it is not only a simulation, but a bisimulation. The existence of a diagonal map ensures that if (i) $\Gamma_1 \vdash_\alpha p$ and $f_{\Gamma_1,\alpha}(p) = q$ and (ii) $\Gamma_2 \vdash_\alpha q'$ with $(\Gamma_1, \alpha) <: (\Gamma_2, \alpha)$ and $q = q'\restriction_{\Gamma_1,\alpha}$ then there is p' such that $f_{\Gamma_2,\alpha}(p') = q'$. In simpler words, the existence of a diagonal map says that if $r(p) = q$ and "q can be extended as $q \trianglelefteq q'$" then "p can be extended accordingly".

The fact that $\sim$ satisfies the rules such as (3) (given in Section 3), can be proved by "proof manipulation". As explained, to show that $P \sim Q$, it suffices to give a functional bisimulation between $[\![P]\!](\Gamma, \alpha)$ and $[\![Q]\!](\Gamma, \alpha)$. As a special case, let us consider the case where $P = (\boldsymbol{\nu}\bar{a}a)(!a(x).P \mid \bar{a}\langle y\rangle)$ and $Q = (\boldsymbol{\nu}\bar{a}a)(!a(x).P \mid P\{y/x\})$. In this case, a functional bisimulation f can be defined by $f(p) \stackrel{\text{def}}{=} q$, where $p \xrightarrow{0} q$. The proof that this f is a bisimulation is similar to that of subject reduction/expansion. For example, if $f(p) = q$ and $q \trianglelefteq q'$ then it suffices to construct a linear process p' such that $p' \xrightarrow{0} q'$ (subject to the condition that p' is a suitable extension of p) as in the proof of subject expansion. Checking that $\sim$ satisfies the other axioms can be done similarly.

We can also show that $\sim$ is a congruence relation. Checking that $\sim$ is a congruence is not that difficult, thanks to the fact that $\trianglelefteq$ is defined according to the structure of a process. Also note that, unlike in the traditional π-calculus, we do not have any problem with input prefixing since communication only occurs between names that are explicitly bound by $\boldsymbol{\nu}$ in the π_F-calculus. That is, placing a process P into a context $C \stackrel{\text{def}}{=} !a(x).[\,]$ does not add new possibilities for interactions among names in P.

[9] To be more specific, we define the open-map bisimulation in the setting where $y\mathcal{E}_\Delta$ (the Yoneda embedding of $\mathcal{E}_\Delta$) is the path category and $[\mathcal{E}^{\text{op}}_\Delta, \mathbf{Sets}]$ is the category of models.

The fact that $\sim$ is a congruence relation implies the following theorem.

▶ **Theorem 19.** *π_F-processes modulo $\sim$ form a compact closed Freyd category.*

The main theorem (Theorem 4), which states the existence of a compact closed Freyd model that is fully abstract with respect to $\simeq^c_\tau$, is a consequence of the above theorem and the following lemma:

▶ **Lemma 20.** *If $P \sim Q$ then $P \simeq^c_\tau Q$.*

This lemma is proved by showing that $\sim$ implies $\overset{\bullet}{\sim}_\tau$ (and using the fact that $\sim$ is a congruence), which basically follows from the relation between linear approximations and $\Rightarrow$ (Proposition 14). Roughly speaking, to show that $P \sim Q$ implies $P \overset{\bullet}{\sim}_\tau Q$ it suffices to show that the following relation is a barbed bisimulation, where $\mathcal{R}$ is the relation used in Proposition 14.

$$\left\{ (P_k, Q_k) \;\middle|\; \begin{array}{l} (P = P_0 \Rightarrow P_1 \Rightarrow \cdots \Rightarrow P_k)\ \mathcal{R}\ p_k \\ (Q = Q_0 \Rightarrow Q_1 \Rightarrow \cdots \Rightarrow Q_k)\ \mathcal{R}\ q_k \\ p_k \sim q_k \\ \text{for some sequences } P = P_0 \Rightarrow P_1 \Rightarrow \cdots \Rightarrow P_k \text{ and} \\ Q = Q_0 \Rightarrow Q_1 \Rightarrow \cdots \Rightarrow Q_k \text{ and} \\ \text{some linear approximations } p_k \text{ and } q_k \end{array} \right\}$$

Here $p \sim q$ means that there exists a span of open maps $[\![P]\!] \xleftarrow{f} X \xrightarrow{g} [\![Q]\!]$ and an element x of $X(\Gamma, \alpha)$ such that $f_{\Gamma,\alpha}(x) = p$ and $g_{\Gamma,\alpha}(x) = q$, i.e. p and q are "bisimilar states". Strictly speaking, we cannot directly use Proposition 14 because P and Q have free outputs. However, the same argument can be applied to the current situation to show the correspondence between $\trianglelefteq$ and extending a reduction sequence, which concludes the above lemma.

7 Related Work

The notion of presheaf plays a central role in our work, but we are not the first one to use presheaf to model the π-calculus. Cattani et al. [8] gave a denotational semantics of the π-calculus within an indexed category of profunctors. The model is fully abstract in the sense that bisimulation in the model, obtained from open maps, coincides with strong late bisimulation. Although their work and our work both use presheaf (or profunctor), they are conceptually different. Our work is motivated by categorical type theory correspondence, whereas the work by Cattani et al. [8] is motivated by a desire to obtain a systematic and algebraic understanding of bisimulation. From a technical point of view, the definition of the path category is significantly different as well. Their path category is indexed by the category of finite name sets and injective maps so that it can treat fresh names as in the domain theoretic models of the π-calculus [20, 11]. On the other hand, our path category is simply the category of type environments of an intersection type system.

A non-idempotent intersection type system for a variant of the π-calculus has also been introduced by Dal Lago et al. [13]. This intersection type system is also inspired by the notion of linear approximation. The connection between linear approximations and intersection types [14] was applied to the encoding of π-calculus to proof-nets to derive the basis of an intersection type system for a fragment of the local π-calculus [24, 16] called *hyperlocalised*

π-*calculus*. They showed that the type system obtained this way characterises some "good behaviour", such as deadlock-freedom, of hyperlocalised processes. In contrast to our work, they use intersection types to guarantee that typable processes are "well-behaved", rather than to define the "operational semantics" of the calculus.

As briefly explained in the introduction, the delays that forwarders add has also been an issue in the field of game semantics. In game semantics, forwarders correspond to copycat strategies and the delay copycat strategies introduce was an obstacle to model synchronous computations using game semantics. Game models in which a "copycat strategy that does not introduce any delay" can be expressed were recently introduced by Castellan and Yoshida to give a fully abstract game semantics of the *synchronous* session π-calculus [7] and by Melliès in a framework called template games [15]. Although these work are apparently different from ours, we believe that they are relevant to our work given that there is a tight relationship between game semantics and linear approximations [22]; detailed comparisons are left for future work.

8 Conclusion

We proposed a variant of the π-calculus whose barbed congruence $\simeq^c_\tau$ can be captured categorically in return for having a non-standard reduction relation $\Rightarrow$. Technically, to handle $\Rightarrow$, we developed a system of linear approximations that captures the behaviour of a process and developed an LTS based on linear approximations. The standard reduction relation $\rightarrow$ and $\Rightarrow$ have been related by the translation $(-)^\dagger$, and we showed that $(P)^\dagger \simeq^c_\tau (Q)^\dagger$ implies $P \simeq^c Q$ ($\simeq^c$ is the conventional barbed congruence). Although we fail to achieve full abstraction, this result is important because it suggests the possibility of using compact closed Freyd models to reason about conventional π-calculus via the translation, which is the future direction we aim to pursue.

References

1 Samson Abramsky. Proofs as processes. *Theor. Comput. Sci.*, 135(1):5–9, 1994. doi:10.1016/0304-3975(94)00103-0.

2 Samson Abramsky, Simon J. Gay, and Rajagopal Nagarajan. Interaction categories and the foundations of typed concurrent programming. In *Proceedings of the NATO Advanced Study Institute on Deductive Program Design, Marktoberdorf, Germany*, pages 35–113, 1996.

3 Gianluigi Bellin and Philip J. Scott. On the π-calculus and linear logic. *Theor. Comput. Sci.*, 135(1):11–65, 1994. doi:10.1016/0304-3975(94)00104-9.

4 Michele Boreale. On the expressiveness of internal mobility in name-passing calculi. *Theor. Comput. Sci.*, 195(2):205–226, 1998. doi:10.1016/S0304-3975(97)00220-X.

5 Gérard Boudol. The lambda-calculus with multiplicities (abstract). In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 1993. doi:10.1007/3-540-57208-2_1.

6 Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *Mathematical Structures in Computer Science*, 26(3):367–423, 2016. doi:10.1017/S0960129514000218.

7 Simon Castellan and Nobuko Yoshida. Two sides of the same coin: session types and game semantics: a synchronous side and an asynchronous side. *PACMPL*, 3(POPL):27:1–27:29, 2019. doi:10.1145/3290340.

8 Gian Luca Cattani, Ian Stark, and Glynn Winskel. Presheaf models for the pi-calculus. In *Category Theory and Computer Science, 7th International Conference, CTCS '97, Santa Margherita Ligure, Italy, September 4-6, 1997, Proceedings*, pages 106–126, 1997. doi:10.1007/BFb0026984.

9 Gian Luca Cattani and Glynn Winskel. Presheaf models for concurrency. In *Computer Science Logic, 10th International Workshop, CSL '96, Annual Conference of the EACSL, Utrecht, The Netherlands, September 21-27, 1996, Selected Papers*, pages 58–75, 1996. doi:10.1007/3-540-63172-0_32.

10 Thomas Ehrhard and Laurent Regnier. Uniformity and the taylor expansion of ordinary lambda-terms. *Theor. Comput. Sci.*, 403(2-3):347–372, 2008. doi:10.1016/j.tcs.2008.06.001.

11 Marcelo P. Fiore, Eugenio Moggi, and Davide Sangiorgi. A fully abstract model for the π-calculus. *Inf. Comput.*, 179(1):76–117, 2002. doi:10.1006/inco.2002.2968.

12 André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation from open maps. *Inf. Comput.*, 127(2):164–185, 1996. doi:10.1006/inco.1996.0057.

13 Ugo Dal Lago, Marc de Visme, Damiano Mazza, and Akira Yoshimizu. Intersection types and runtime errors in the pi-calculus. *PACMPL*, 3(POPL):7:1–7:29, 2019. doi:10.1145/3290320.

14 Damiano Mazza, Luc Pellissier, and Pierre Vial. Polyadic approximations, fibrations and intersection types. *PACMPL*, 2(POPL):6:1–6:28, 2018. doi:10.1145/3158094.

15 Paul-André Melliès. Categorical combinatorics of scheduling and synchronization in game semantics. *PACMPL*, 3(POPL):23:1–23:30, 2019. doi:10.1145/3290336.

16 Massimo Merro. *Locality in the π-calculus and applications to distributed objects*. PhD thesis, École Nationale Supérieure des Mines de Paris, 2000.

17 Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996. doi:10.1017/S096012950007002X.

18 John Power and Edmund Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5):453–468, 1997. doi:10.1017/S0960129597002375.

19 Ken Sakayori and Takeshi Tsukada. A categorical model of an **i**/**o**-typed π-calculus. In *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, pages 640–667, 2019. doi:10.1007/978-3-030-17184-1_23.

20 Ian Stark. A fully abstract domain model for the π-calculus. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 36–42, 1996. doi:10.1109/LICS.1996.561301.

21 Takeshi Tsukada, Kazuyuki Asada, and C.-H. Luke Ong. Generalised species of rigid resource terms. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005093.

22 Takeshi Tsukada and C.-H. Luke Ong. Plays as resource terms via non-idempotent intersection types. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 237–246, 2016. doi:10.1145/2933575.2934553.

23 Glynn Winskel and Mogens Nielsen. Presheaves as transition systems. In *Partial Order Methods in Verification, Proceedings of a DIMACS Workshop, Princeton, New Jersey, USA, July 24-26, 1996*, pages 129–140, 1996. doi:10.1090/dimacs/029/08.

24 Nobuko Yoshida. Minimality and separation results on asynchronous mobile processes – representability theorems by concurrent combinators. *Theor. Comput. Sci.*, 274(1-2):231–276, 2002. doi:10.1016/S0304-3975(00)00310-8.

A Compact closed Freyd category and π_F-theory

We briefly review the correspondence between the π_F-calculus and compact closed Freyd category originally proposed in [19] to make this paper self-contained.

▶ **Definition 21** (Compact closed Freyd category [19]). *A* compact closed Freyd category *is a Freyd category [18] $J\colon \mathcal{C} \to \mathcal{K}$ such that (1) $\mathcal{K}$ is compact closed, and (2) J has the (chosen) right adjoint $I \Rightarrow (-)\colon \mathcal{K} \to \mathcal{C}$.*

An equivalence $\mathcal{E}$ is a *π_F-theory* if it is closed under the following rules. Each rule has implicit assumptions that the both sides of the equation are well-sorted processes.

$$\frac{a \notin \mathbf{fn}(P, C) \qquad \bar{a} \notin \mathbf{bn}(C)}{\Gamma \vdash (\boldsymbol{\nu} \bar{a}a)(!a(\vec{x}).P \mid C[\bar{a}\langle \vec{y} \rangle]) = (\boldsymbol{\nu} \bar{a}a)(!a(\vec{x}).P \mid C[P\{\vec{y}/\vec{x}\}])} \quad \text{(E-Beta)}$$

$$\frac{a, \bar{a} \notin \mathbf{fn}(P)}{\Gamma \vdash (\boldsymbol{\nu} \bar{a}a)!a(\vec{y}).P = \mathbf{0}} \text{ (E-GC)} \qquad \frac{\bar{a}, a \notin \mathbf{fn}(\bar{c}\langle \vec{x} \rangle)}{\Gamma \vdash \bar{c}\langle \vec{x} \rangle = (\boldsymbol{\nu} \bar{a}a)(a \hookrightarrow \bar{b} \mid \bar{c}\langle \vec{x}\{\bar{a}/\bar{b}\} \rangle)} \text{ (E-FOut)}$$

$$\frac{b, \bar{a} \notin \mathbf{fn}(P)}{\Gamma \vdash (\boldsymbol{\nu} \bar{a}a)(b \hookrightarrow \bar{a} \mid P) = P\{b/a\}} \quad \text{(E-Eta)}$$

$$\frac{P \equiv Q}{\Gamma \vdash P = Q} \text{ (E-SCong)} \qquad \frac{\Delta \vdash P = Q \qquad C\colon \Gamma/\Delta\text{-context}}{\Gamma \vdash C[P] = C[Q]} \text{ (E-Ctx)}$$

Here a (Γ/Δ)-context is a context C such that $\Gamma \vdash C[P]$ for every $\Delta \vdash P$.

Any set Ax of equations-in-context has the minimum theory $Th(Ax)$ that contains Ax. We write $Ax \rhd \Delta \vdash P = Q$ if $(\Delta \vdash P = Q) \in Th(Ax)$. It should be noted that the original paper [19] only considers theory over the empty signature and that the π_F-calculus over the empty signature does not have the constant τ. The calculus defined in this paper is a π_F-calculus defined over the signature with a single constant $\tau : \mathbf{ch}^i[]$.

The important property that has been used in the body of this paper is that the term model $Cl(Ax)$ is a compact closed Freyd category for every set of non-logical axioms Ax [19, Theorem 3]. Given a set Ax of non-logical axioms, the term model $Cl(Ax)$ is defined as processes modulo $Ax \rhd \Delta \vdash P = Q$. Objects are list of types and a morphism (of the compact closed category) from $\vec{T}$ to $\vec{S}$ is an equivalence class $[\vec{x} : \vec{T}, \vec{y} : \vec{S}^{\perp} \vdash P]$. The composition of morphisms is defined by "parallel composition + hiding". For morphisms $P : \vec{T} \to \vec{S}$ and $Q : \vec{S} \to \vec{U}$, i.e. processes such that $\vec{x} : \vec{T}, \vec{y} : \vec{S}^{\perp} \vdash P$ and $\vec{z} : \vec{S}, \vec{w} : \vec{U}^{\perp} \vdash Q$, their composite is $\vec{x} : \vec{T}, \vec{w} : \vec{U}^{\perp} \vdash (\boldsymbol{\nu} \vec{y}\vec{z})(P|Q)$. (See [19] for the full definition.)

B Supplementary materials for Section 4

B.1 Groupoid structure of types and type isomorphisms

As expected, the witness of type isomorphisms can be *composed* so that $\varphi_1\colon \rho_1 \cong \rho_2$ and $\varphi_2\colon \rho_2 \cong \rho_3$ implies $(\varphi_2 \circ \varphi_1)\colon \rho_1 \cong \rho_3$. Composition of witnesses are defined by:

$$\mathbf{ch}^o_\alpha[\varphi'_1, \ldots, \varphi'_n] \circ \mathbf{ch}^o_\alpha[\varphi_1, \ldots, \varphi_n] \stackrel{\text{def}}{=} \mathbf{ch}^o_\alpha[\varphi_1 \circ \varphi'_1, \ldots, \varphi_n \circ \varphi'_n]$$

$$\mathbf{ch}^i_\alpha[\varphi'_1, \ldots, \varphi'_n] \circ \mathbf{ch}^i_\alpha[\varphi_1, \ldots, \varphi_n] \stackrel{\text{def}}{=} \mathbf{ch}^i_\alpha[\varphi'_1 \circ \varphi_1, \ldots, \varphi'_n \circ \varphi_n]$$

$$(\sigma_2, (\varphi'_i)_{i \in \mathbf{Nat}}) \circ (\sigma_1, (\varphi_i)_{i \in \mathbf{Nat}}) \stackrel{\text{def}}{=} (\sigma_2\sigma_1, (\varphi'_{\sigma_1(i)} \circ \varphi_i)_{i \in \mathbf{Nat}})$$

Types and type isomorphisms forms a groupoid. That is, we can define the inverse operator $(-)^{-1}$ for witnesses of type isomorphisms and show that there is an identity $\mathrm{id}_\rho \colon \rho \cong \rho$ for every type ρ. The inverse operator $(-)^{-1}$ is defined by $(\mathbf{ch}^m_\alpha[\varphi_1, \ldots, \varphi_n])^{-1} \stackrel{\text{def}}{=} \mathbf{ch}^m_\alpha[\varphi_1^{-1}, \ldots, \varphi_n^{-1}]$ for $m \in \{i, o\}$ and $(\sigma, (\varphi_i)_{i \in \mathbf{Nat}})^{-1} \stackrel{\text{def}}{=} (\sigma^{-1}, (\varphi^{-1}_{\sigma^{-1}(i)})_{i \in \mathbf{Nat}})$.

B.2 Subject reduction/expansion and Proposition 11

This section outlines the proof of Proposition 11, which states the correspondence between execution sequences and linear approximations. The proposition is a consequence of the subject reduction/expansion lemma. The proof for Proposition 14 (which we omit) is similar; the only difference is that we also need to take the order $\trianglelefteq$ into account.

As usual, to prove the subject reduction we use a substitution lemma:

▶ **Lemma 22** (Substitution Lemma). *Suppose that* $\Gamma \sqcap x : (i, \rho) \vdash_\alpha p$, $\varphi \colon \rho' \cong \rho$ *and* $\Gamma \sqcap y : (j, \rho')$ *is defined. Then* $\Gamma \sqcap y : (j, \rho') \vdash_\alpha p\{\varphi \cdot y_j / x_i\}$.

The proof of this lemma is similar to that of the conventional substitution lemma, except for the fact that we need to take group actions into account. Similarly, the following lemma can be proved by induction on the structure of p.

▶ **Lemma 23.** *Let* $\Gamma \sqcap x : (i, \rho) \vdash_\alpha p$, $\varphi \colon \rho' \cong \rho$ *and assume that* $y_j \notin \mathbf{fn}(p)$. *Then* $p\{\varphi \cdot y_j / x_i\}\{\varphi^{-1} \cdot x_i / y_j\} = p$

Now the subject reduction/expansion lemmas, and similar lemmas for the τ-reduction can be stated as follows. We omit the proofs as they can be shown by standard arguments with the help of Lemma 22 and 23.

▶ **Lemma 24** (Subject reduction). *Assume that* $\Gamma \vdash_\alpha p$ *and* $p \xrightarrow{0} q$. *Then we have* $\Gamma \vdash_\alpha q$. *Moreover, if* $\Gamma^\natural \vdash p \sqsubset P$ *then there exists* Q *such that* $\Gamma^\natural \vdash q \sqsubset Q$ *and* $P \xrightarrow{0} Q$.

▶ **Lemma 25.** *Suppose that* $\Gamma \sqcap \tau : (i, \mathbf{ch}^i_\beta[]) \vdash_\alpha p$, $\beta \le \gamma$ *for all* $\gamma \in \overline{\mathbf{lv}}(\Gamma)$ *and* $p \xrightarrow{\tau_i} q$. *Then we have* $\Gamma \vdash_\beta q$. *Moreover, if* $(\natural\Gamma) \sqcap \tau : \{i\} \vdash p \sqsubset P$, *then there exists* Q *such that* $\Gamma^\natural \vdash q \sqsubset Q$ *and* $P \xrightarrow{\tau} Q$.

▶ **Lemma 26** (Subject expansion). *Suppose that* $P \xrightarrow{0} P'$, $\Gamma \vdash_\alpha p'$ *and* $\Gamma^\natural \vdash p' \sqsubset P'$. *Then there exists* p *such that* $\Gamma \vdash_\alpha p$, $\Gamma^\natural \vdash p \sqsubset P$ *and* $p \xrightarrow{0} p'$.

▶ **Lemma 27.** *Suppose that* $P \xrightarrow{\tau} Q$, $\Gamma \vdash_\alpha q$ *and* $\Gamma^\natural \vdash q \sqsubset Q$. *For all* $i \notin \mathrm{dom}(\Gamma(\tau))$, *there exists* p *and* β *such that* $\Gamma \sqcap \tau : (i, \mathbf{ch}^i_\beta[]) \vdash_\alpha p$, $\Gamma^\natural \sqcap \tau : \{i\} \vdash p \sqsubset P$ *and* $p \xrightarrow{\tau_i} q$.

Now we are ready to prove Proposition 11.

▶ **Proposition 11.** *Let* $\tau : \mathbf{ch}^i[] \vdash P$, *i.e. let* P *be a process without any free names.*

1. *Suppose that* $\Gamma \vdash_\alpha p$ *and* $\Gamma^\natural \vdash p \sqsubset P$. *If* $p \xrightarrow{\pi} q$ *then we have* $(p \xrightarrow{\pi} q) \sqsubset (P \xrightarrow{\pi} Q)$ *for some* Q.
2. *Assume* $P \xrightarrow{\pi} Q$, $\Gamma \vdash_\alpha q$ *and* $\Gamma^\natural \vdash q \sqsubset Q$. *Then we have* $(p \xrightarrow{\pi} q) \sqsubset (P \xrightarrow{\pi} Q)$ *for some* p.

Proof. (Proof of 1.) Let us write $\mathbf{ch}^i_{\alpha_i}[]$ for $\Gamma(\tau)(i)$ when $i \in \mathrm{dom}(\Gamma(\tau))$. By the assumption that $p \xrightarrow{\pi} q$, there exists a sequence $p = p_0 \xrightarrow{l_1} \cdots \xrightarrow{l_n} p_n = q$. Let $\tau_{i_1} \ldots \tau_{i_k}$ be the subword of π that is obtained by deleting 0 from π. Without loss of generality, we may assume that $\alpha_{i_1} < \cdots < \alpha_{i_k}$ and $\alpha_{i_k} < \alpha$ for all $\alpha \in \{\alpha_i \mid i \in \mathrm{dom}(\Gamma(\tau))\} \setminus \{\alpha_{i_1}, \ldots, \alpha_{i_k}\}$;

if not we can always reannotate the levels appearing in Γ and use that type environment instead of Γ. Now suppose that $l_1 = \tau_{i_1}$. Then we can apply Lemma 25 to obtain P_1 such that $P_0 \xrightarrow{\tau} P_1$ and $\Gamma_1^\natural \vdash p_1 \sqsubset P_1$, where Γ_1 is the type environment that satisfy $\Gamma_1 \vdash_{\alpha_{i_1}} p_1$. If $l_1 = 0$ we can use Lemma 24 instead. By repeating this argument we obtain a sequence $P = P_0 \xrightarrow{l_1} \cdots \xrightarrow{l_n} P_n = Q$ that can be used to show $(p \xrightarrow{\pi} q) \sqsubset (P \xrightarrow{\pi} Q)$.

(Proof of 2.) Since $P \xrightarrow{\pi} Q$, we have $P = P_0 \xrightarrow{l_1} \cdots \xrightarrow{l_n} P_n = Q$, where $\pi = l_1 \dots l_n$. Let us consider the case where $l_n = \tau$. In this case we can appeal to Lemma 27 (if $l_n = 0$ we use Lemma 26). By Lemma 27, we have p_{n-1} such that (1) $p_{n-1} \xrightarrow{\tau} q$, (2) $\Gamma \sqcap \tau : (i, \mathbf{ch}^i_\beta[]) \vdash_\beta p_{n-1}$ and (3) $\Gamma^\natural \sqcap \tau : \{i\} \vdash p_{n-1} \sqsubset P_{n-1}$, for some index i such that $i \notin \mathrm{dom}(\Gamma(\tau))$ and some level β. By repeating the argument we obtain $p \xrightarrow{\pi} q$ with the desired property. ◀

C Supplementary materials for Section 5

C.1 Restriction of types and type isomorphisms

▶ **Definition 28** (Complete version of Definition 12)**.**
Let A be a set of levels. Restriction of types and intersection types *are inductively defined by:*

$$\mathbf{ch}^m_\alpha[\theta_1, \dots, \theta_n]\restriction_A \stackrel{\text{def}}{=} \begin{cases} \mathbf{ch}^m_\alpha[\theta_1\restriction_A, \dots, \theta_n\restriction_A] & (\textit{if } \alpha \in A) \\ \bullet & (\textit{otherwise}) \end{cases}$$

$$(\theta\restriction_A)(i) \stackrel{\text{def}}{=} \theta(i)\restriction_A,$$

where $m \in \{i, o\}$.

Similarly, restriction of type isomorphisms *is defined by:*

$$\mathbf{ch}^m_\alpha[\varphi_1, \dots, \varphi_n] \stackrel{\text{def}}{=} \begin{cases} \mathbf{ch}^m_\alpha[\varphi_1\restriction_A, \dots, \varphi_n\restriction_A] & (\textit{if } \alpha \in A) \\ \mathrm{id}_\bullet & (\textit{otherwise}) \end{cases}$$

$$(\sigma, (\varphi_i)_{i \in \mathbf{Nat}})\restriction_A \stackrel{\text{def}}{=} (\sigma, (\varphi_i\restriction_A)_{i \in \mathbf{Nat}})$$

where $m \in \{i, o\}$. We write $\rho' <: \rho$ (resp. $\theta' <: \theta$) if $\rho' = \rho\restriction_A$ (resp. $\theta' = \theta\restriction_A$) for some $A \subseteq \mathcal{A}$ and $\varphi' <: \varphi$ if $\varphi' = \varphi\restriction_A$ for some $A \subseteq \mathcal{A}$.

C.2 Overview for the proof of Theorem 17

Here we briefly explain how to show that $[\![P]\!](-,-)$ is a presheaf (Theorem 17). Since Theorem 17, which says that $[\![P]\!]$ is a presheaf, is an immediate consequence of Proposition 16, the main goal of this section is to sketch the proof of Proposition 16:

▶ **Proposition 16.** *Assume that $\Gamma \vdash_\alpha p$, p is terminable and $(\Gamma', \alpha) <: (\Gamma, \alpha)$. Then there is a unique (up to $\equiv_0$) linear process that satisfy $q \trianglelefteq p$ and $\Gamma' \vdash_\alpha q$.*

The proof of Proposition 16 proceeds by induction on the structure of the derivation of $\Gamma \vdash_\alpha p$. The non-trivial case is the case of ν-restriction because it is not clear how the type annotated to the $\boldsymbol{\nu}$ binder should be restricted. To handle this case, we use the following lemmas, which says that "how the annotated type should be restricted is determined by how the type environment is restricted".

▶ **Lemma 29.** *Suppose that $\Gamma \vdash_\alpha (\boldsymbol{\nu}[\langle \bar{a}_i, a_i \rangle]_{i \in \mathrm{dom}(\theta)})p$ and that $(\boldsymbol{\nu}[\langle \bar{a}_i, a_i \rangle]_{i \in \mathrm{dom}(\theta)})p$ is terminable. Then $\overline{\mathbf{lv}}(\theta) \subseteq \overline{\mathbf{lv}}(\Gamma) \cup \{\alpha\}$.*

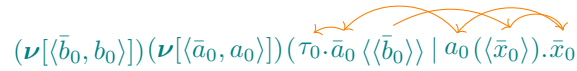
▶ **Lemma 30.** *Let* $(\boldsymbol{\nu}[\langle \bar{a}_i, a_i\rangle]_{i\in \mathrm{dom}(\theta)})p$ *be a terminable process typed under* Γ*, i.e.* $\Gamma \vdash_\alpha (\boldsymbol{\nu}[\langle \bar{a}_i, a_i\rangle]_{i\in \mathrm{dom}(\theta)})p$*. Suppose that* $(\boldsymbol{\nu}[\langle \bar{a}_i, a_i\rangle]_{i\in \mathrm{dom}(\theta')})q \trianglelefteq (\boldsymbol{\nu}[\langle \bar{a}_i, a_i\rangle]_{i\in \mathrm{dom}(\theta)})p$ *and* $\Gamma' \vdash_\alpha (\boldsymbol{\nu}[\langle \bar{a}_i, a_i\rangle]_{i\in \mathrm{dom}(\theta')})q$*, where* Γ' *satisfies* $\Gamma'(t)(i) <: \Gamma(t)(i)$ *for all term* t *and index* i*.If there is a level* β *such that* $\beta \in \overline{\mathbf{lv}}(\theta)$ *but* $\beta \notin \overline{\mathbf{lv}}(\theta')$*, then there is a term* t *and an index* i *such that* $\beta \in \Gamma(t)(i)$ *and* $\beta \notin \Gamma'(t)(i)$*.*

Instead of giving a detailed proof of these lemmas, we look at an example.

▶ **Example 31.** Let us consider a well typed linear process

$$\tau : (0, \mathbf{ch}^i_\beta[]) \vdash_\gamma (\boldsymbol{\nu}[\langle \bar{b}_0, b_0\rangle_{\rho_b}])(\boldsymbol{\nu}[\langle \bar{a}_0, a_0\rangle_{\rho_a}])(\tau_0.\bar{a}_0\langle\langle b_0\rangle\rangle \mid a_0(\langle \bar{x}_0\rangle).\bar{x}_0\langle\langle\rangle\rangle \mid b_0(\langle\rangle))$$

where $\rho_b = \mathbf{ch}^o_\beta[]$ and $\rho_a = \mathbf{ch}^o_\alpha[(0, \rho_\beta)]$. The following figure shows the way to point a free name (or a constant τ_i) whose type contains the level $\beta \in \overline{\mathbf{lv}}(\rho_b)$. (In this case we can tell that the type for τ_0 contains β.)



Let us explain what the pointers mean. A pointer points to a name that must be "executed at the same time" with the name placed at the source of the pointer. We start from $\bar{b}_0$ because that is the name with type ρ_b. Since $\bar{b}_0$ is in an object position of an output via the name $\bar{a}_0$ and $\bar{a}_0$ is bound, we first look for the name that communicates with $\bar{a}_0$, which is a_0 in this case. Because $\bar{x}_0$ is the argument that corresponds to $\bar{b}_0$, the type for $\bar{x}_0$ must have the level β for its "outermost level". So now we have another name $\bar{x}_0$ whose type has β as the "outermost level", and the link from $\bar{b}_0$ to $\bar{x}_0$ is used to expresses this fact. Now we look for the place where $\bar{x}_0$ is actually used, this is expressed by the second link. Since $\bar{x}_0$ is guarded by a_0 we now know that a_0 must be executed at the same time as $\bar{x}_0$. Because $\bar{a}_0$ communicates with a_0, we know that $\bar{a}_0$ and a_0 must be executed simultaneously and thus we have a pointer from a_0 to $\bar{a}_0$. The output $\bar{a}_0$ is guarded by τ_0, so we know that τ_0 also happens at the same time. Because τ_0 is a constant we conclude that β appears in the type environment.

Lemma 29 can be proved by formalising the notion of pointer and generalising the above procedure.

Lemma 30 can be proved by showing that $\trianglelefteq$ does not create any "dangling pointer". That is if $q \trianglelefteq p$ and linear terms t_j, t_i appearing in p are linked by a pointer, then either t_j and t_i both appears in q or t_j and t_i do not appear in q. This follows from the definition of $\trianglelefteq$ and the way we add pointers. For example, let us consider the case where p is the linear process depicted above. The only process q such that $\bar{b}_0$ does not appear in q and $q \trianglelefteq p$ is $(\boldsymbol{\nu}[])(\boldsymbol{\nu}[])(\bot \mid \bot \mid \bot)$.

With the above lemmas it is straightforward to prove Proposition 16 by induction on the structure of the derivation of $\Gamma \vdash_\alpha p$ and Theorem 17 follows as a corollary of this proposition.