A Categorical Model of an i/o-typed π-calculus

Ken Sakayori and Takeshi Tsukada

University of Tokyo, Japan

Abstract. This paper introduces a new categorical structure that is a model of a variant of the **i**/**o**-typed π-calculus, in the same way that a cartesian closed category is a model of the λ-calculus. To the best of our knowledge, no categorical model has been given for the **i**/**o**-typed π-calculus, in contrast to session-typed calculi, to which corresponding logic and categorical structure were given. The categorical structure introduced in this paper has a simple definition, combining two well-known structures, namely, closed Freyd category and compact closed category. The former is a model of effectful computation in a general setting, and the latter describes connections via channels, which cause the effect we focus on in this paper. To demonstrate the relevance of the categorical model, we show by a semantic consideration that the π-calculus is equivalent to a core calculus of Concurrent ML.

Keywords: π-calculus, categorical type theory, compact closed category, closed Freyd category

1 Introduction

The Curry-Howard-Lambek correspondence reveals the trinity of the simply-typed λ-calculus, propositional intuitionistic logic and cartesian closed category. Via the correspondence, a type of the calculus can be seen as a formula of the logic, and as an object of a category; a term can be seen as a proof and as a morphism (see, e.g., [23]). Since its discovery, a number of variations have been proposed and studied.

In concurrency theory, a correspondence between a process calculus and logic was established by Caires, Pfenning and Toninho [8,9] and later by Wadler [48]. What they found is that session types [18,20] can be seen as formulas of linear logic [14], and processes as proofs. This remarkable result has inspired lots of work (e.g. [45,46,25,3,4,10]).

This correspondence is, however, not completely satisfactory as pointed out in [3,26], as well as by Wadler himself [48]. The session-typed calculi in [9,48] corresponding to linear logic have only well-behaved processes, because the session type systems guarantee deadlock-freedom and race-freedom of well-typed processes. This strong guarantee is often useful for programmers writing processes in the typed calculus, but can be seen as a significant limitation of expressive power. For example, it prevents us from modelling wild concurrent systems or programs that might fall into deadlocks or race conditions.

This paper describes an approach to a Curry-Howard-Lambek correspondence for concurrency in the presence of deadlocks and race conditions, from the viewpoint of categorical type theory.

What is the categorical model of the π-calculus? We focus on the π-calculus [30,31] in this paper. This is not only because the π-calculus is widely used and powerful, but also because of a classical result by Sangiorgi [39,42], which is the starting point of our development.

Sangiorgi, in the early 90s, gave translations between the conventional, first-order π-calculus and its higher-order variant [39,42]. This translation allows us to regard the π-calculus as a higher-order programming language.

Let us review the observation by Sangiorgi, using a core of the asynchronous π-calculus: $P ::= \mathbf{0} \mid (P|Q) \mid \bar{a}\langle x\rangle \mid a(x).P$.[1] The idea is to decompose the input-prefixing $a(x).P$ into a and $(x).P$. Let us write $a[(x).P]$ for $a(x).P$ to emphasise the decomposition. Then a reduction can also be decomposed as

$$\bar{a}\langle x\rangle \mid a[(y).P] \mid Q \longrightarrow [(y).P]\langle x\rangle \mid Q \longrightarrow P\{x/y\} \mid Q,$$

where the first step is the communication and the second step is the β-reduction (i.e. $(\lambda y.P)\, x \longrightarrow P\{x/y\}$ in the λ-calculus notation). Hence we regard

- an output $\bar{a}\langle x\rangle$ as an application of a function $\bar{a}$ to x, and
- an input $a(x).P$ as an abstraction $(x).P$ (or $\lambda x.P$) "located" at $a[-]$.

Now, ignoring the mysterious operator $a[-]$, what we had are the core operations of functional programming languages (i.e. abstraction and application). This functional programming language is effectful; in fact, communication via channels is a side effect.

This observation leads us to base our categorical model for the π-calculus on a model for effectful functional programs. Among several models, we choose *closed Freyd category* [37] for modelling the functional part.

Then what is the categorical counterpart of $a[-]$? As this operation seems responsible for communication, this question can be rephrased as: what is the categorical structure for communication? An observation by Abramsky et al. [2] answered this question. They pointed out the importance of *compact closed category* [21] in concurrency theory, which nicely describes CCS-like processes interconnected via ports.

By combining the two structures described above, this paper introduces a categorical structure, which we call *compact closed Freyd category*, as a categorical model of the π-calculus.[2] Despite its simplicity, compact closed Freyd

[1] This calculus slightly differs from the calculus we shall introduce in Section 2, but the differences are not important here.

[2] Here is the reason why we do not use a monad for modelling the effect: it is unclear for us how to integrate a monad with the compact closed structure. On the contrary, a Freyd category has a (pre)monoidal category as its component; we can simply require that it is compact closed.

category captures the strong expressive power of the π-calculus. The compact closed structure allows us to connect ports in an arbitrary way, in return for the possibility of deadlocks; the Freyd structure allows us to duplicate objects, and duplication of input channels introduces the possibility of race conditions.

Reconstructing calculi This paper introduces two calculi that are sound and complete with respect to the compact closed Freyd category model. One is a variant of the π-calculus, named π_F; the design of π_F is based on the observations described above. The other is a higher-order programming language λ_{ch} defined as an instance of the computational λ-calculus [33]. Designing λ_{ch} is not so difficult because we can make use of the correspondence between computational λ-calculus and closed Freyd category (see Section 4). The λ_{ch}-calculus have operations for creating a channel and for sending a value via the channel and, therefore, can be seen as a core calculus of *Concurrent ML* (or *CML*) [38].

Since the higher-order calculus λ_{ch} and π_F correspond to the same categorical model, we can obtain translations between these calculi by simple semantic computations. These translations are "correct by definition" and, interestingly, coincide with those between higher-order and first-order π-calculus [39,42].

On β- vs. $\beta\eta$-theories The categorical analysis of this paper reveals that many conventional behavioural equivalences for the π-calculus are problematic from a viewpoint of categorical type theory. The problem is that they induce only *semicategories*, which may not have identities for some objects. This is a reminiscent of the β-theory of the λ-calculus, of which categorical model is given by semi-categorical notions [16].

Adding a single rule (which we call the *η-rule*) resolves the problem. Our categorical type theory deals with only equivalences that admits the η-rule, and the simplicity of the theory of this paper essentially relies on the η-rule.

Interestingly the η-rule seems to explain some phenomenon in the literature. For example, Sangiorgi observed that a syntactic constraint called *locality* [28,49] is essential for his translation [39,42]. The correctness of the translation can be proved without using the η-rule, when one restricts the calculus local; we expect that Sangiorgi's observation can be related to this phenomenon.

Contributions This paper introduces a new variant of the **i**/**o**-typed π-calculus, which we call π_F. A remarkable feature of π_F is that it has a categorical counterpart, called compact closed Freyd category. The correspondence is fairly firm; the categorical semantics is sound and complete, and the term model is the classifying category. The relevance of the model is demonstrated by a semantic reconstruction of Sangiorgi's translation [39,42]. These results open a new frontier in the Curry-Howard-Lambek correspondence for concurrency; session-type is not the only base for a Curry-Howard-Lambek correspondence for π-calculi.

Organisation of this paper. Section 2 introduces the calculus π_F and discuss equivalences on processes. Section 3 gives the categorical semantics of π_F and

shows soundness and completeness. A connection to a higher-order programming language with channels is studied in Section 4. In Section 5, we (1) discuss how our work relates to linear logic and (2) present some ideas for how to extend the application range of our model. We discuss related work in Section 6 and conclude in Section 7. Omitted proofs, as well as detailed definitions, are available in the full version.

2 A polyadic, asynchronous π-calculus with i/o-types

This section introduces a variant of π-calculus, named π_F. It is based on a fairly standard calculus, namely polyadic and asynchronous π-calculus with **i**/**o**-types, but the details are carefully designed so that π_F has a categorical model.

2.1 The π_F-calculus

This subsection defines the calculus π_F, which is based on an asynchronous variant of the polyadic π-calculus with **i**/**o**-types in [35]. The aim of this subsection is to explain what are the differences from the conventional π-calculus. Although π_F has some uncommon features, each of them was studied in the literature; see Related Work (Section 6) for related ideas and calculi.

Types The set of *types*, 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 type $\mathbf{ch}^o[T_1, \ldots, T_n]$ is for output channels sending n arguments of types $T_1, \ldots, T_n$. The type $\mathbf{ch}^i[T_1, \ldots, T_n]$ is for input channels. The *dual* $T^\perp$ of type T is defined by $\mathbf{ch}^o[\vec{T}]^\perp \stackrel{\text{def}}{=} \mathbf{ch}^i[\vec{T}]$ and $\mathbf{ch}^i[\vec{T}]^\perp \stackrel{\text{def}}{=} \mathbf{ch}^o[\vec{T}]$. For a sequence $\vec{T} \stackrel{\text{def}}{=} T_1, \ldots, T_n$ of types, we write $\vec{T}^\perp$ for $T_1^\perp, \ldots, T_n^\perp$.

An important difference from [35] is that no channel allows both input and output operations. We will refer this feature of π_F as **i**/**o**-*separation*.

Processes Let $\mathcal{N}$ be a denumerable set of *names*, ranged over by x, y and z. Each name is either input-only or output-only, because of **i**/**o**-separation.

The set of *processes*, ranged over by P, Q and R, is defined by

$$P, Q, R \ ::= \ \mathbf{0} \mid (P|Q) \mid (\boldsymbol{\nu}_{\mathbf{ch}^o[\vec{T}]}\, xy)P \mid x\langle\vec{y}\rangle \mid !x(\vec{y}).P \quad .$$

The notion of *free names*, as well as *bound names*, is defined as usual. The set of free names (resp. bound names) of P is written as $\mathbf{fn}(P)$ (resp. $\mathbf{bn}(P)$). We allow tacit renaming of bound names, and identify α-equivalent processes.

The meaning of the constructs should be clear, except for $(\boldsymbol{\nu}_T\, xy)P$ which is less common. The process $\mathbf{0}$ is the inaction; $P \mid Q$ is a parallel composition; $x\langle\vec{y}\rangle$ is an output; and $!x(\vec{x}).P$ is a replicated input. The restriction $(\boldsymbol{\nu}_T\, xy)P$

$$\frac{}{\Gamma \vdash \mathbf{0} : \diamond} \qquad \frac{\Gamma \vdash P : \diamond \qquad \Gamma \vdash Q : \diamond}{\Gamma \vdash P \mid Q : \diamond} \qquad \frac{\Gamma, x : \mathbf{ch}^o[\vec{T}], y : \mathbf{ch}^i[\vec{T}] \vdash P : \diamond}{\Gamma \vdash (\boldsymbol{\nu}_{\mathbf{ch}^o[\vec{T}]}\, xy)P : \diamond}$$

$$\frac{(x : \mathbf{ch}^i[\vec{T}]) \in \Gamma \qquad \Gamma, \vec{y} : \vec{T} \vdash P : \diamond}{\Gamma \vdash !x(\vec{y}).P : \diamond} \qquad \frac{(x : \mathbf{ch}^o[\vec{T}]) \in \Gamma \qquad \vec{y} : \vec{T} \subseteq \Gamma}{\Gamma \vdash x\langle \vec{y} \rangle : \diamond}$$

Fig. 1. Typing rules for processes

hides the names x and y of type T and $T^\perp$ and, at the same time, establishes a connection between x and y. Communication takes place only over bound names explicitly connected by $\boldsymbol{\nu}$. This is in contrast to the conventional π-calculus, in which input-output correspondence is *a priori* (i.e. $\bar{a}$ is the output to a).

The π_F-calculus does not have non-replicated input $x(\vec{y}).P$.

Typing rules A *type environment* Γ is a finite sequence of type bindings of the form $x\colon T$. We assume the names in Γ are pairwise distinct. If $\vec{x} = x_1, \ldots, x_n$ and $\vec{T} = T_1, \ldots, T_n$, we write $\vec{x}\colon \vec{T}$ for $x_1\colon T_1, \ldots, x_n\colon T_n$. We write $(\vec{x}\colon \vec{T}) \subseteq \Gamma$ to mean $x_i\colon T_i \in \Gamma$ for every i.

A *type judgement* is of the form $\Gamma \vdash P : \diamond$, meaning that P is a well-typed process under Γ. The typing rules are listed in Fig. 1.

Notation 1. We define $(\boldsymbol{\nu}_{\mathbf{ch}^i[\vec{T}]}\, xy)P$ as $(\boldsymbol{\nu}_{\mathbf{ch}^o[\vec{T}]}\, yx)P$; then $(\boldsymbol{\nu}_T\, xy)P$ is defined for every T. We abbreviate $(\boldsymbol{\nu}_{T_1}\, x_1 y_1) \ldots (\boldsymbol{\nu}_{T_n}\, x_n y_n)P$ as $(\boldsymbol{\nu}_{\vec{T}}\, \vec{x}\vec{y})P$. We often omit type annotations and write $(\boldsymbol{\nu} xy)$ for $(\boldsymbol{\nu}_T\, xy)$ and $(\boldsymbol{\nu}\vec{x}\vec{y})$ for $(\boldsymbol{\nu}_{\vec{T}}\, \vec{x}\vec{y})$. We use a and b for names of input channel types and $\bar{a}$ and $\bar{b}$ for output. Note that a and $\bar{a}$ are connected only if they are bound by the same occurrence of $\boldsymbol{\nu}$. □

Operational semantics *Structural congruence*, written $\equiv$, is the smallest congruence relation on processes that satisfies the following rules:

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

where $x, y \notin \mathbf{fn}(Q)$ in the fourth rule and w, x, y, z are distinct in the fifth rule.

The *reduction relation* on processes, written $\longrightarrow$, is defined by the base rule

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

(where $P\{\vec{x}/\vec{y}\}$ is the capture-avoiding substitution) and the structural rule which concludes $P \longrightarrow Q$ from $\exists P'\, Q'.\ P \equiv P' \longrightarrow Q' \equiv Q$. Note that, unlike conventional π-calculi, communication only occurs over bound names connected by ν. We write $\longrightarrow^*$ for the reflexive and transitive closure of $\longrightarrow$.

It should be clear that deadlocks and racy communications can be expressed in π_F. An example of race is $(\boldsymbol{\nu}\bar{a}a)(\bar{a}\langle \vec{y} \rangle \mid !a(\vec{x}).P \mid !a(\vec{x}).Q)$, where two input actions are trying to consume the output regarded as a resource. A similar process $(\boldsymbol{\nu}\bar{a}a)(!a(\vec{x}).P \,|\, \bar{a}\langle \vec{y} \rangle \mid \bar{a}\langle \vec{z} \rangle)$ does not have a race since the receiver $!a(\vec{x}).P$ is replicated. In general, race conditions on output actions do not occur in π_F.

2.2 Equivalences on processes

To establish a Curry-Howard-Lambek correspondence is to find a nice algebraic or categorical structure of terms. For example, the original Curry-Howard-Lambek correspondence reveals the cartesian closed structure of λ-terms.

Such a nice structure would become visible only when appropriate notions of composition and of equivalence could be identified, such as substitution and $\beta\eta$-equivalence for the λ-calculus.

As for process calculi, so-called "parallel composition + hiding" paradigm [17] has been used to compose processes. Given typed processes

$$\vec{x} : \vec{T},\ \vec{y} : \vec{S} \vdash P : \diamond \quad \text{and} \quad \vec{w} : \vec{S}^{\perp},\ \vec{u} : \vec{U} \vdash Q : \diamond,$$

their composite via $(\vec{y}, \vec{w})$ is defined as

$$\vec{x} : \vec{T},\ \vec{u} : \vec{U} \vdash (\boldsymbol{\nu}_{\vec{S}}\, \vec{y}\vec{w})(P \mid Q) : \diamond.$$

This kind of composition appears quite often in logical studies of π-calculi [1,5,19]. It also plays a central role in *interaction category paradigm* proposed by Abramsky, Gay and Nagarajan [2].

So it remains to determine an equivalence on π-calculus processes, appropriate for our purpose. This subsection approaches the problem from two directions:

- Examining behavioural equivalences proposed and studied in the literature
- Developing a new equivalence based on categorical considerations

Let us clarify the notion of equivalence discussed below. An *equation-in-context* is a judgement of the form $\Gamma \vdash P = Q$, where $\Gamma \vdash P : \diamond$ and $\Gamma \vdash Q : \diamond$. An *equivalence* $\mathcal{E}$ is a set of equations-in-context that is reflexive, transitive and symmetric (e.g. $(\Gamma \vdash P = P) \in \mathcal{E}$ for every $\Gamma \vdash P : \diamond$).

Behavioural equivalences As mentioned above, we are interested in the structure of π_F-processes modulo existing behavioural equivalences. Among the various behavioural equivalence, we start with studying *barbed congruence* [32], which is one of the most widely used equivalences.

We define (asynchronous and weak) barbed congruence for π_F. For each name $\bar{a}$, we write $P{\downarrow_{\bar{a}}}$ if $P \equiv (\boldsymbol{\nu}\vec{x}\vec{y})(\bar{a}\langle\vec{z}\rangle \mid Q)$ and $\bar{a}$ is free, and $P{\Downarrow_{\bar{a}}}$ if $\exists Q.\, P \longrightarrow^{*} Q{\downarrow_{\bar{a}}}$. A (Γ/Δ)-*context* is a context C such that $\Gamma \vdash C[P] : \diamond$ for every $\Delta \vdash P : \diamond$.

Definition 1. *A* barbed bisimulation *is a symmetric relation $\mathcal{R}$ on processes such that, whenever $P\ \mathcal{R}\ Q$, (1) $P{\downarrow_{\bar{a}}}$ implies $Q{\Downarrow_{\bar{a}}}$ and (2) $P \longrightarrow P'$ implies $\exists Q'.\, (Q \longrightarrow^{*} Q') \wedge (P'\ \mathcal{R}\ Q')$.* Barbed bisimilarity $\overset{\bullet}{\approx}$ *is the largest barbed bisimulation. Typed processes $\Delta \vdash P : \diamond$ and $\Delta \vdash Q : \diamond$ are* barbed congruent at Δ, *written $\Delta \vdash P \cong^{c} Q$, if $C[P] \overset{\bullet}{\approx} C[Q]$ for every (Γ/Δ)-context C.* □

Let us consider a category-like structure $\mathcal{C}$ in which an object is a type and a morphism is an equivalence class of π_F-processes modulo barbed congruence. More precisely, a morphism from T to S is a process $x\colon T,\ y\colon S^{\perp} \vdash P\colon \diamond$ modulo

barbed congruence (and renaming of free names x and y). Then the composition (i.e. "parallel composition + hiding") is well-defined on equivalence classes, because barbed congruence is a congruence. This is a fairly natural setting.

We have a strikingly negative result.

Theorem 1. *$\mathcal{C}$ is not a category.*

Proof. In every category, if $f : A \longrightarrow A$ is a left-identity on A (i.e. $f \circ g = g$ for every $g : A \longrightarrow A$), then f is the identity on A. The process $a : \mathbf{ch}^o[], \bar{b} : \mathbf{ch}^i[] \vdash !a().\bar{b}\langle\rangle : \diamond$ seen as a morphism $(\mathbf{ch}^o[]) \longrightarrow (\mathbf{ch}^o[])$ is a left-identity but not the identity. The former means that $c\colon \mathbf{ch}^o[], \bar{b}\colon \mathbf{ch}^i[] \vdash \big((\boldsymbol{\nu}\bar{a}a)(!a().\bar{b}\langle\rangle \mid P)\big) \mathrel{\approx^c} P\{\bar{b}/\bar{a}\}$ for every $c\colon \mathbf{ch}^o[], \bar{a}\colon \mathbf{ch}^i[] \vdash P\colon \diamond$, which is a consequence of the *replicator theorems* [35]. To prove the latter, observe that $(\boldsymbol{\nu}\bar{b}b)(!a().\bar{b}\langle\rangle \mid \mathbf{0})$ and $\mathbf{0}$ are not barbed congruent. Indeed the context $C \stackrel{\text{def}}{=} (\boldsymbol{\nu}\bar{a}a)(\bar{a}\langle\rangle \mid !a().\bar{o}\langle\rangle \mid [\,])$ distinguishes the processes, where $\bar{o}$ is the observable. □

Note that race condition is essential for the proof, specifically, for the part proving that the process $!a().\bar{b}\langle\rangle$ is not the identity. A race condition occurs in $C[(\boldsymbol{\nu}\bar{b}b)(!a().\bar{b}\langle\rangle \mid \mathbf{0})]$, where $\bar{a}$ in C has two receivers.

The process $!a().\bar{b}\langle\rangle$ is called *forwarder*, and forwarders will play a central role in this paper. Its general form is $a \hookrightarrow \bar{b} \stackrel{\text{def}}{=} !a(\vec{x}).\bar{b}\langle\vec{x}\rangle$. When $x : T$ and $y : T^{\perp}$, we write $x \leftharpoondown y$ to mean $x \hookrightarrow y$ if $T = \mathbf{ch}^i[\vec{S}]$ and otherwise $y \hookrightarrow x$.

Remark 1. The argument in the proof of Theorem 1 is widely applicable to **i**/**o**-typed calculi, not specific to π_F. In particular, **i**/**o**-separation (i.e. absence of $\mathbf{ch}^{i/o}[\vec{T}]$) is not the cause, but the existence of $\mathbf{ch}^o[\vec{T}]$ or $\mathbf{ch}^i[\vec{T}]$ is. □

Remark 2. Session-typed calculi in Caires, Pfenning and Toninho [8,9], which correspond to linear logic, do not seem to suffer from this problem. In our understanding, this is because of race-freedom of their calculi. □

To obtain a category, we should think of a coarser equivalence that identifies $(\boldsymbol{\nu}\bar{b}b)(!a().\bar{b}\langle\rangle \mid \mathbf{0})$ with $\mathbf{0}$. Such an equivalence should be very coarse; even *must-testing equivalence* [11] fails to equate them. As long as we have checked, only *may-testing equivalence* [11] defined below satisfies the requirement.

Definition 2. *Typed processes $\Delta \vdash P : \diamond$ and $\Delta \vdash Q : \diamond$ are* may-testing equivalent *at Δ, written $\Delta \vdash P =_{may} Q$, if $C[P]\Downarrow_{\bar{a}} \Leftrightarrow C[Q]\Downarrow_{\bar{a}}$ for every (Γ/Δ)-context C and name $\bar{a}$.* □

As we shall see, π_F-processes modulo may-testing equivalence behaves well. May-testing equivalence is, however, often too coarse.

Category-driven approach In this approach, we first guess an appropriate categorical structure sufficient for interpreting π_F, based on intuitions discussed in Introduction (see also Section 3.1), and then design an equivalence so that it is sound and complete with respect to the categorical semantics.

Figure 2 defines the equivalence, described as a set of rules. A *π_F-theory* is an equivalence that behaves well from the categorical perspective.

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

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

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

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

Fig. 2. Inference rules of equations-in-context. Each rule has implicit assumptions that the both sides of the equation are well-typed processes.

Definition 3. *An equivalence $\mathcal{E}$ is a π_F-*theory *if it is closed under the rules in Fig. 2. Any set Ax of equations-in-context has the minimum theory $Th(Ax)$ that contains Ax. We write $Ax \rhd \Gamma \vdash P = Q$ if $(\Gamma \vdash P = Q) \in Th(Ax)$.* □

Let us examine each rule in Fig. 2.

The rule (E-Beta) should be compared with the reduction relation. When $C = ([\,] \mid Q)$, then (E-Beta) claims

$$(\boldsymbol{\nu} \bar{a}a)(!a(\vec{x}).P \mid \bar{a}\langle \vec{y} \rangle \mid Q) \quad = \quad (\boldsymbol{\nu} \bar{a}a)(!a(\vec{x}).P \mid P\{\vec{y}/\vec{x}\} \mid Q)$$

provided that $a \notin \mathbf{fn}(P, Q)$, which is indeed an instance of the reduction.

A significant difference from reduction is the side condition. It is essential in the presence of race conditions. Without the side condition, every π_F-theory would be forced to contain the symmetric and transitive closure of the reduction relation; thus it would identify $P \mid (\boldsymbol{\nu} \bar{a}a)(!a().P \mid !a().Q)$ with $Q \mid (\boldsymbol{\nu} \bar{a}a)(!a().P \mid !a().Q)$ for every processes P and Q (where $\bar{a}, a$ are fresh), because

$$\begin{aligned} (\boldsymbol{\nu} \bar{a}a)(\bar{a}\langle\rangle \mid !a().P \mid !a().Q) \quad &\longrightarrow \quad P \mid (\boldsymbol{\nu} \bar{a}a)(!a().P \mid !a().Q) \\ (\boldsymbol{\nu} \bar{a}a)(\bar{a}\langle\rangle \mid !a().P \mid !a().Q) \quad &\longrightarrow \quad Q \mid (\boldsymbol{\nu} \bar{a}a)(!a().P \mid !a().Q). \end{aligned}$$

The side condition prevents π_F-theories from collapsing.

Another, relatively minor, difference is that application of (E-Beta) is not limited to the contexts of the form $[\,] \mid Q$. This kind of extension can be found in, for example, work by Honda and Laurent [19] studying π-calculus from a logical perspective.

The rule (E-GC) runs "garbage-collection". Because no one can send a message to the hidden name a, the process $!a(\vec{x}).P$ will never be invoked and thus is safely discarded. This rule is sound with respect to many behavioural equivalences, including barbed congruence. Rules of this kind often appear in the literature studying logical aspects of concurrent calculi (as in Honda and Laurent [19] and Wadler [48]). There is, however, a subtle difference in the side condition: (E-GC) requires that a and $\bar{a}$ do not appear at all in P.

The rule (E-FOut) can be seen as the η-rule of abstractions, as in the λ-calculus and in the higher-order π-calculus [39]. In the latter, an output name $\bar{b}$ can be identified with an abstraction $(\vec{y}).\bar{b}\langle\vec{y}\rangle$. Then we have, for example,

$$(\boldsymbol{\nu}\bar{a}a)(a \hookrightarrow \bar{b} \mid \bar{c}\langle\bar{a}\rangle) \;=\; (\boldsymbol{\nu}\bar{a}a)(a \hookrightarrow \bar{b} \mid \bar{c}\langle\,(\vec{y}).\bar{a}\langle\vec{y}\rangle\,\rangle) \;=\; \bar{c}\langle\,(\vec{y}).\bar{b}\langle\vec{y}\rangle\,\rangle \;=\; \bar{c}\langle\bar{b}\rangle$$

where we use (E-Beta) and (E-GC) in the second step. An important usage of (E-FOut) is to replace an output of free names with that of bound names. This kind of operation has been studied in [7,28] as a part of translations from the π-calculus to its local/internal fragments.[3]

The rule (E-Eta) requires the forwarders are left-identities, directly describing the requirement discussed above.[4]

The rules (E-SCong) and (E-Ctx) are easy to understand. The former requires that structurally congruent processes should be identified; the latter says that a π_F-theory is a congruence.

These rules can be justified from the operational viewpoint, as well. A well-known result on the **i**/**o**-typed π-calculus (see, e.g., [43,35]) shows the following propositions.

Proposition 1. *Barbed congruence is closed under all rules but* (E-Eta). □

Proposition 2. *May-testing equivalence is a π_F-theory.* □

In particular, the latter means that may-testing equivalence is in the scope of the categorical framework of this paper; see Theorem 5.

3 Categorical semantics

This section introduces the class of *compact closed Freyd categories* and discusses the interpretation of the π_F-calculus in the categories. We show that the categorical semantics is sound and complete with respect to the equational theory given in Section 2.2, and that the syntax of the π_F-calculus induces a model.

This section, by its nature, is slightly theoretical compared with other sections. Section 3.1 explains the ideas of this section without heavily using categorical notions; the subsequent subsections require familiarity with categorical type theory.

3.1 Overview

As mentioned in Section 1, the categorical model of π_F is *compact closed Freyd category*, which has both closed Freyd and compact closed structures. Here we

[3] Free outputs can be eliminated from π_F-processes by using the rules (E-FOut) and (E-Eta), i.e. external mobility can be encoded by internal mobility [7,40]. If the calculus is local [28,49], then we do not need (E-Eta) to eliminate free outputs.

[4] A forwarder behaves as a right-identity with respect to every π_F-theory. This is a consequence of rules (E-Beta), (E-GC) and (E-FOut).

informally discuss what is a compact closed Freyd category and how to interpret π_F by using syntactic representation.

A *closed Freyd category* is a model of higher-order programs with side effects. It has, among others, the structures to interpret the function type $A \Rightarrow B$ and its constructor and destructor, namely, abstraction $\lambda x.t$ and application $t\,u$. It also has a mechanism for unrestricted duplication of variables; in terms of logic, contraction is admissible.

A *compact closed category* can be seen as MLL [14] with the left rule:

$$\frac{\Gamma, A^*, A \vdash I}{\Gamma \vdash I} \qquad \left[\frac{\Gamma \vdash A^* \qquad \Delta \vdash A}{\Gamma, \Delta \vdash I}\right].$$

(The right rule is the companion, which itself is derivable in MLL.)

A *compact closed Freyd category* has all the constructs. It has the structures corresponding to the following type constructors:

$$\text{(closed Freyd)} \quad I, A \otimes B, A \Rightarrow B \qquad \text{(compact closed)} \quad I, A \otimes B, A^*.$$

Note that the pair type $A \otimes B$ (as well as the unit I) coming from the closed Freyd structure is identified with that from the compact closed structure. Inference rules for a compact closed Freyd category is those for functional languages and the above rules of the compact closed structure.

Interpreting π_F in a compact closed Freyd category is to interpret it by using these constructs. As mentioned in Section 1, following Sangiorgi [39], we regard

- an output $\bar{a}\langle\vec{x}\rangle$ as an application of a function $\bar{a}$ to a tuple $\langle\vec{x}\rangle$, and
- an input $!a(\vec{x}).P$ as an abstraction $(\vec{x}).P$ (or $\lambda\vec{x}.P$) located at a.

We interpret the output action by using the function application. Hence the type $\mathbf{ch}^o[T]$ is regarded as a function type $T \Rightarrow I$ (where the unit type I is the type for processes i.e. $\diamond$); then the typing rule for output actions becomes

$$\frac{\Gamma, \bar{a}\colon (T \Rightarrow I), x\colon T \vdash \bar{a} : T \Rightarrow I \qquad \Gamma, \bar{a}\colon (T \Rightarrow I), x\colon T \vdash x : T}{\Gamma, \bar{a}\colon (T \Rightarrow I), x\colon T \vdash \bar{a}\langle x\rangle : I}$$

The type $\mathbf{ch}^i[T]$ is understood as $(T \Rightarrow I)^*$; the input-prefixing rule becomes

$$\frac{\dfrac{}{\Gamma, a\colon (T \Rightarrow I)^* \vdash a : (T \Rightarrow I)^*} \qquad \dfrac{\Gamma, a\colon (T \Rightarrow I)^*, x\colon T \vdash P : I}{\Gamma, a\colon (T \Rightarrow I)^* \vdash (x).P : T \Rightarrow I}}{\Gamma, a\colon (T \Rightarrow I)^* \vdash !a(x).P : I}$$

This derivation directly expresses the intuition that an input-prefixing is abstraction followed by allocation; here allocation is interpreted by using the compact closed structure, i.e. connection of ports. The name restriction also has a natural derivation:

$$\frac{\Gamma, a\colon (T \Rightarrow I)^*, \bar{a}\colon (T \Rightarrow I) \vdash P : I}{\Gamma \vdash (\boldsymbol{\nu}\bar{a}a)P : I}$$

3.2 Compact closed Freyd category

Let us formalise the ideas given in Section 3.1. Hereafter in this section, we assume basic knowledge of category theory and of categorical type theory.

We recall the definitions of compact closed category and closed Freyd category. For simplicity, the structures below are strict and chosen; a functor is required to preserve the chosen structures on the nose.

Definition 4 (Compact closed category [21]). *Let $(\mathcal{C}, \otimes, I)$ be a symmetric strict monoidal category. The* dual *of an object A in $\mathcal{C}$ is an object A^* equipped with* unit $\eta_A \colon I \longrightarrow A \otimes A^*$ *and* counit $\epsilon_A \colon A^* \otimes A \longrightarrow I$ *that satisfy the "triangle identities"* $(\eta_A \otimes \mathrm{id}_A); (\mathrm{id}_A \otimes \epsilon_A) = \mathrm{id}_A$ *and* $(\mathrm{id}_{A^*} \otimes \eta_A); (\epsilon_A \otimes \mathrm{id}_{A^*}) = \mathrm{id}_{A^*}$. *The category $\mathcal{C}$ is* compact closed *if each object is equipped with a chosen dual.* □

Definition 5 (Closed Freyd category [37]). *A* Freyd category *is given by (1) a category with chosen finite products $(\mathcal{C}, \otimes, I)$, called* value category*, (2) a symmetric strict monoidal category $(\mathcal{K}, \otimes, I, \mathbf{symm})$, called* producer category*, and (3) an identity-on-object strict symmetric monoidal functor $J \colon \mathcal{C} \to \mathcal{K}$. A Freyd category is a* closed Freyd category *if the functor $J(-) \otimes A \colon \mathcal{C} \to \mathcal{K}$ has the (chosen) right adjoint $A \Rightarrow - \colon \mathcal{K} \to \mathcal{C}$ for every object A. We write $\Lambda_{A,B,C}$ for the natural bijection $\mathcal{K}(J(A) \otimes B, C) \longrightarrow \mathcal{C}(A, B \Rightarrow C)$ and $\mathbf{eval}_{A,B}$ for $\Lambda^{-1}(\mathrm{id}_{A \Rightarrow B}) \colon (A \Rightarrow B) \otimes A \longrightarrow B$ in $\mathcal{K}$.* □

Remark 3. The above definition is a restriction of the original one [37], in which $\mathcal{K}$ is a *premonoidal* [36] category. This change reflects concurrency of the calculus. In fact, it validates the following law, expressed by the syntax of the computational λ-calculus [33],

$$\mathbf{let}\, x = M \,\mathbf{in}\, \mathbf{let}\, y = N \,\mathbf{in}\, L \qquad = \qquad \mathbf{let}\, y = N \,\mathbf{in}\, \mathbf{let}\, x = M \,\mathbf{in}\, L.$$

Then one can evaluate M by using the left form and N by using the right form. This law allows us to evaluate M and N in arbitrary order, or concurrently. □

We now introduce the categorical structure corresponding to the π_F-calculus.

Definition 6 (Compact closed Freyd category). *A* compact closed Freyd category *is a Freyd category $J : \mathcal{C} \longrightarrow \mathcal{K}$ such that (1) $\mathcal{K}$ is compact closed, and (2) J has the (chosen) right adjoint $I \Rightarrow - \colon \mathcal{K} \to \mathcal{C}$.* □

We shall often write J for a compact closed Freyd category $J \colon \mathcal{C} \overset{\perp}{\circlearrowright} \mathcal{K}$.

A compact closed Freyd category is a closed Freyd category:

$$\mathcal{K}(J(A) \otimes B, C) \;\cong\; \mathcal{K}(J(A), B^* \otimes C) \;\cong\; \mathcal{C}(A, I \Rightarrow (B^* \otimes C)).$$

Example 1. The most basic example of a compact closed Freyd category is (the strict monoidal version of) $J \colon \mathbf{Sets} \overset{\perp}{\circlearrowright} \mathbf{Rel} \colon \mathcal{P}$. Here J is the identity-on-object functor that maps a function to its graph and $\mathcal{P}$ is the "power set functor" that maps a relation $\mathcal{R} \subseteq A \times B$ to a function $\mathcal{P}(\mathcal{R}) \overset{\text{def}}{=} \{(S_A, S_B) \mid S_B = \{b \mid a \in S_A, a\, \mathcal{R}\, b\}\}$. Another example is obtained by replacing sets with posets, functions with monotone functions and relations with downward closed relations. □

$$
\begin{aligned}
[\![\mathbf{ch}^i[T_1,\dots,T_n]]\!] &\stackrel{\text{def}}{=} (([\![T_1]\!]\otimes\cdots\otimes[\![T_n]\!])\Rightarrow I)^*\\
[\![\mathbf{ch}^o[T_1,\dots,T_n]]\!] &\stackrel{\text{def}}{=} ([\![T_1]\!]\otimes\cdots\otimes[\![T_n]\!])\Rightarrow I\\
\\
[\![\Gamma\vdash \mathbf{0}:\diamond]\!] &\stackrel{\text{def}}{=} J(!_\Gamma)\\
[\![\Gamma\vdash !a(\vec{x}).P:\diamond]\!] &\stackrel{\text{def}}{=} J(\langle \pi_a^\Gamma, \Lambda_{\Gamma,\vec{T},I}([\![\Gamma,\vec{x}:\vec{T}\vdash P:\diamond]\!])\rangle);\epsilon_{\mathbf{ch}[\vec{T}]}\\
[\![\Gamma\vdash \bar{a}\langle\vec{x}\rangle:\diamond]\!] &\stackrel{\text{def}}{=} J(\langle \pi_{\bar{a}}^\Gamma, \pi_{x_1}^\Gamma,\dots,\pi_{x_n}^\Gamma\rangle);\mathbf{eval}_{\vec{T},I}\\
[\![\Gamma\vdash P\mid Q:\diamond]\!] &\stackrel{\text{def}}{=} J(\Delta_\Gamma);([\![\Gamma\vdash P:\diamond]\!]\otimes[\![\Gamma\vdash Q:\diamond]\!])\\
[\![\Gamma\vdash (\boldsymbol{\nu} xy)P:\diamond]\!] &\stackrel{\text{def}}{=} (\mathrm{id}_\Gamma\otimes\eta_T);[\![\Gamma,x:T,y:T^\perp\vdash P:\diamond]\!]
\end{aligned}
$$

Fig. 3. Interpretation of types and processes. Here $!_\Gamma$, Δ_Γ and π_y^Γ are maps in $\mathcal{C}$ induced by the cartesian structure, namely, $!_\Gamma\colon [\![\Gamma]\!]\longrightarrow I$ is the terminal map, $\Delta_\Gamma\colon [\![\Gamma]\!]\longrightarrow [\![\Gamma]\!]\otimes[\![\Gamma]\!]$ is the diagonal map and, when $\Gamma=(y_1\colon T_1,\dots,y_n\colon T_n)$ and $x=y_j$, the morphism $\pi_x^\Gamma\colon [\![\Gamma]\!]\longrightarrow [\![T_j]\!]$ is the j-th projection. The interpretation of a type environment $x_1\colon T_1,\dots,x_n\colon T_n$ is $[\![T_1]\!]\otimes\cdots\otimes[\![T_n]\!]$.

Example 2. A more sophisticated example is taken from Laird's game-semantic model of π-calculus [22]. Precisely speaking, the model in [22] itself is not compact closed Freyd, but its variant (with non-negative arenas) is. This model is important since it is fully abstract w.r.t. may-testing equivalence [22, Theorem 1]; hence our framework has a model that captures the may-testing equivalence. □

3.3 Interpretation

Given a compact closed Freyd category $J\colon \mathcal{C}\overset{\perp}{\circlearrowright}\mathcal{K}$, this section defines the interpretation $[\![-]\!]_J$. It maps types and type environments to objects as usual, and a well-typed process $\Gamma\vdash P:\diamond$ to a morphism $[\![P]\!]\colon [\![\Gamma]\!]\to I$ in $\mathcal{K}$ (recall that the tensor unit I is the interpretation of the type for processes).

Figure 3 defines the interpretation of types and processes. It simply formalises the ideas presented in Section 3.1: for example, the interpretation of $!a(\vec{x}).P$ is the abstraction Λ (from the closed Freyd structure) followed by location ϵ (from the compact closed structure). There are some points worth noting.

- $(A\Rightarrow I)^*$ is *not* isomorphic to $A^*\Rightarrow I$, $A\Rightarrow I$ nor $I\Rightarrow A$. Indeed $(A\Rightarrow I)^*$ cannot be simplified. Do not confuse it with a valid law $I\Rightarrow(A^*)\cong A\Rightarrow I$.
- A parallel composition is interpreted as a pair. Recall that two components of a pair are evaluated in parallel in this setting (cf. Remark 3).
- All but the last rule use the cartesian structure of $\mathcal{C}$ in order to duplicate or discard the environment.

Example 3. Let us consider $y:T\vdash(\boldsymbol{\nu}\bar{a}a)(\bar{a}\langle y\rangle\mid !a(x).P):\diamond$, where $\bar{a},a,y\notin\mathbf{fn}(P)$ and $a\colon \mathbf{ch}^i[T]$. By (E-Beta) and (E-GC), this process is equal to $P\{y/x\}$. It is natural to expect that the interpretations of the two processes coincide; indeed it is. As the following calculation indicates, our semantics factorises the

reduction into two steps: (1) the "transmission" of the closure $\lambda\vec{x}.P$ by the triangle identity of the compact closed structure, and (2) the β-reduction modelled by **eval** of the closed Freyd structure:

$$
\begin{aligned}
&[\![y : T \vdash (\boldsymbol{\nu}\bar{a}a)(\bar{a}\langle y\rangle \mid !a(x).P) : \diamond]\!] \\
&= (\mathrm{id}_T \otimes \eta_{\mathbf{ch}^o[T]}); [\![y : T, \bar{a} : \mathbf{ch}^o[T], a : \mathbf{ch}^i[T] \vdash \bar{a}\langle y\rangle \mid !a(x).P : \diamond]\!] \\
&= (\mathrm{id} \otimes \eta); ([\![y : T, \bar{a} : \mathbf{ch}^o[T] \vdash \bar{a}\langle y\rangle : \diamond]\!] \otimes [\![a : \mathbf{ch}^i[T] \vdash !a(x).P : \diamond]\!]) \\
&= (\mathrm{id} \otimes \eta); ((\mathbf{symm}_{T,\mathbf{ch}^o[T]}; \mathbf{eval}_{T,I}) \otimes (\mathrm{id}_{\mathbf{ch}[T]^*} \otimes J(\Lambda([\![x : T \vdash P : \diamond]\!])))); \epsilon_{T \Rightarrow I} \\
&= (\mathrm{id}_T \otimes J(\Lambda([\![x : T \vdash P : \diamond]\!]))); \mathbf{symm}_{T,\mathbf{ch}^o[T]}; \mathbf{eval}_{T,I} \quad \text{(By triangle identity)} \\
&= (J(\Lambda([\![x : T \vdash P : \diamond]\!])) \otimes \mathrm{id}_T); \mathbf{eval}_{T,I} \\
&= [\![x : T \vdash P]\!] \qquad \text{(By the universality of } \mathbf{eval}\text{)} \\
&= [\![y : T \vdash P\{y/x\} : \diamond]\!].
\end{aligned}
$$

(Here we implicitly use derived rules for weakening and exchange.) □

Example 4. The interpretation of a forwarder $a : \mathbf{ch}^i[\vec{T}], \bar{b} : \mathbf{ch}^o[\vec{T}] \vdash a \hookrightarrow \bar{b} : \diamond$ is the counit $\epsilon_{\mathbf{ch}^o[\vec{T}]} : [\![\mathbf{ch}^o[\vec{T}]]\!]^* \otimes [\![\mathbf{ch}^o[\vec{T}]]\!] \longrightarrow I$ in $\mathcal{K}$, which is the one-sided form of the identity. Recall that a forwarder is the identity in every π_F-theory. □

The semantics is sound and complete. That means, a judgement $Ax \rhd \Gamma \vdash P = Q$ is provable if and only if $\Gamma \vdash P = Q$ is valid in all models J of Ax.

Here we define the related notions and prove soundness; completeness is the topic of the next subsection.

Definition 7. *An equational judgement $\Gamma \vdash P = Q$ is* valid in J *if $[\![\Gamma \vdash P : \diamond]\!]_J = [\![\Gamma \vdash Q : \diamond]\!]_J$. Given a set Ax of non-logical axioms, J is a* model of Ax*, written $J \models Ax$, if it validates all judgements in Ax. We write $Ax \rhd \Gamma \Vdash P = Q$ if $\Gamma \vdash P = Q$ is valid in every J such that $J \models Ax$.* □

Theorem 2 (Soundness). *If $Ax \rhd \Gamma \vdash P = Q$, then $Ax \rhd \Gamma \Vdash P = Q$.* □

3.4 Term model

A *term model* is a category whose objects are type environments and whose morphisms are terms (i.e. processes in this setting). This section gives a construction of the term model, by which we show completeness. This subsection basically follows the standard arguments in categorical type theory; we mainly focus on the features unique to our model, giving a sketch to the common part.

Given a set Ax of axioms, we define the term model $J_{Ax} : \mathcal{C}_{Ax} \overset{\perp}{\underset{\leftarrow}{\circlearrowright}} \mathcal{K}_{Ax}$, which we also write as $Cl(Ax)$.

The definition of the producer category $\mathcal{K}_{Ax}$ follows the standard recipe. As usual, its objects are finite lists of types. The monoidal product $\vec{T} \otimes \vec{S}$ is the concatenation of the lists and the dual $\vec{T}^*$ is $\vec{T}^\perp$. Given objects $\vec{T}$ and $\vec{S}$, a morphism from $\vec{T}$ to $\vec{S}$ is a process $\vec{x}{:}\ \vec{T}, \vec{y}{:}\ \vec{S}^\perp \vdash P : \diamond$ (modulo renaming of variables $\vec{x}$ and $\vec{y}$). If $Ax \rhd \vec{x}{:}\ \vec{T}, \vec{y}{:}\ \vec{S}^\perp \vdash P = Q$ is provable, then P and

Q are regarded as the same morphism. Composition of morphisms is defined as "parallel composition plus hiding": For morphisms $P : \vec{T} \longrightarrow \vec{S}$ and $Q : \vec{S} \longrightarrow \vec{U}$, i.e. processes such that $\vec{x}\colon \vec{T},\ \vec{y}\colon \vec{S}^{\perp} \vdash P : \diamond$ and $\vec{z}\colon \vec{S},\ \vec{w}\colon \vec{U}^{\perp} \vdash Q : \diamond$, their composite is $\vec{x}\colon \vec{T}, \vec{w}\colon \vec{U}^{\perp} \vdash (\boldsymbol{\nu}\vec{y}\vec{z})(P \mid Q) : \diamond$. The monoidal product $P \otimes Q$ of morphisms is the parallel composition $P \mid Q$. The identity, as well as the symmetry of the monoidal product and the unit and counit of the compact closed structure, is a parallel composition of forwarders: for example, the identity on $\vec{S}$ is $\vec{x} : \vec{S}, \vec{y} : \vec{S}^{\perp} \vdash x_1 \leftrightharpoons y_1 \mid \cdots \mid x_n \leftrightharpoons y_n : \diamond$ where n is the length of $\vec{S}$. The facts that most structural morphisms are forwarders and that forwarders compose are the keys to show that $\mathcal{K}_{Ax}$ is a compact closed category.

We then see the definition of $\mathcal{C}_{Ax}$, of which the definition of morphisms has a subtle point. The objects of $\mathcal{C}_{Ax}$ are by definition the same as $\mathcal{K}_{Ax}$, i.e. lists of types. The definition of morphisms relies on the notion of *values*. The values are defined by the grammar $V ::= x \mid (\vec{x}).P$, where P is a process and $(\vec{x}).P$ is called an *abstraction*. Typing rules for values are as follows:

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \qquad \frac{\Gamma, \vec{x}\colon \vec{T} \vdash P}{\Gamma \vdash (\vec{x}).P : \mathbf{ch}^{o}[\vec{T}]}.$$

(To understand the right rule, recall that $[\![\mathbf{ch}^{o}[\vec{T}]]\!] = [\![\vec{T}]\!] \Rightarrow I$.) A morphism from $\vec{T}$ to $\vec{S} = (S_1, \ldots, S_n)$ is an n-tuple $(V_1, \ldots, V_n)$ of values of type $\vec{x}\colon \vec{T} \vdash V_i\colon S_i$ for each i (modulo renaming of $\vec{x}$). Composition is intuitively defined by "substitution followed by β-reduction" whose definition is omitted here.[5]

The functor J_{Ax} places the values to the channels. For example, let $\vec{T} = (\mathbf{ch}^{i}[U_1], \mathbf{ch}^{o}[U_2])$ and consider the morphism in $\mathcal{C}_{Ax}$ given by

$$a\colon \mathbf{ch}^{i}[T_1], \bar{b}\colon \mathbf{ch}^{o}[T_2] \vdash (a, \bar{b}, (\vec{x}).P) : (\mathbf{ch}^{i}[T_1], \mathbf{ch}^{o}[T_2], \mathbf{ch}^{o}[\vec{S}])$$

where $\vec{S}$ is the type for $\vec{x}$. The image of this morphism by the functor J_{Ax} is

$$a\colon \mathbf{ch}^{i}[T_1], \bar{b}\colon \mathbf{ch}^{o}[T_2], \bar{c}\colon \mathbf{ch}^{o}[T_1], d\colon \mathbf{ch}^{i}[T_2], e\colon \mathbf{ch}^{i}[\vec{S}] \vdash a \hookrightarrow \bar{c} \mid d \hookrightarrow \bar{b} \mid !e(\vec{x}).P\colon \diamond.$$

This example contains all the three ways to place a value to a given channel.

Theorem 3. *$Cl(Ax)$ is a compact closed Freyd category for every Ax.* □

In the model $Cl(Ax)$, the interpretation of a process $\Gamma \vdash P : \diamond$ is the equivalence class that P belongs to. This fact leads to completeness.

Theorem 4 (Completeness). *If $Ax \rhd \Gamma \Vdash P = Q$, then $Ax \rhd \Gamma \vdash P = Q$.* □

Theorem 5. *There exists a compact closed Freyd category J that is fully abstract w.r.t. may-testing equivalence, i.e. $\Gamma \vdash P =_{may} Q$ iff $[\![P]\!]_J = [\![Q]\!]_J$.*

Proof. Let J be the term model $Cl(=_{may})$ and use Proposition 2. □

[5] Here is a subtle technical issue that we shall not address in this paper; see the long version for the formal definition. We think, however, that this paragraph conveys a precise intuition.

3.5 Theory/model correspondence

It is natural to expect that $Cl(Ax)$ is the *classifying category* as in the standard categorical type theory. This means, to give a model of Ax in J is equivalent to give a structure-preserving functor $Cl(Ax) \longrightarrow J$. This subsection clarifies and studies this claim.

The set $\mathrm{Mod}(Ax, J)$ of models of Ax in J is defined as follows. If $J \models Ax$, then $\mathrm{Mod}(Ax, J)$ is a singleton set[6]; otherwise $\mathrm{Mod}(Ax, J)$ is the empty set.

We then define the notion of structure-preserving functors.

Definition 8. *A* strict compact closed Freyd functor *from* $J\colon \mathcal{C} \circlearrowright_{\perp} \mathcal{K}\colon I \Rightarrow (-)$ *to* $J'\colon \mathcal{C}' \circlearrowright_{\perp} \mathcal{K}'\colon I \Rightarrow' (-)$ *is a pair of functor* (Φ, Ψ) *such that*

- Φ *is a strict finite product preserving functor from* $\mathcal{C}$ *to* $\mathcal{C}'$*,*
- Ψ *is a strict symmetric monoidal functor from* $\mathcal{K}$ *to* $\mathcal{K}'$ *that preserves the chosen compact closed structures (i.e. units and counits) on the nose, and*
- (Φ, Ψ) *is a map of adjoints between* $J \dashv I \Rightarrow (-)$ *and* $J' \dashv I \Rightarrow' (-)$*.*

□

The collection of (small) compact closed Freyd categories and strict compact closed Freyd functors form a 1-category, which we write as $CCFC$.

Now the question is whether $\mathrm{Mod}(Ax, J) \stackrel{?}{\cong} CCFC(Cl(Ax), J)$ in **Set**.

Unfortunately this does not hold. More precisely, the left-to-right inclusion does not hold in general. This means that the term model satisfies some additional axioms reflecting some aspects of the π_F-calculus.

The additional axioms reflect the definition of the dual $\vec{T}^*$ in the term model; we have $\vec{T}^* \stackrel{\mathrm{def}}{=} \vec{T}^{\perp}$ by definition, and thus $\vec{T}^{**} = \vec{T}$ and $(\vec{T} \otimes \vec{S})^* = \vec{T}^* \otimes \vec{S}^*$. It might be surprising that these equations are harmful because isomorphisms $A^{**} \cong A$ and $(A \otimes B)^* \cong A^* \otimes B^*$ exist in every compact closed category. The point is that the equations also require $\mathcal{C}$ to have isomorphisms $A^{**} \cong A$ and $(A \otimes B)^* \cong A^* \otimes B^*$ (witnessed by the respective identities).

We formally define the additional axioms, which we call **(I)** and **(D)**:

(I) The canonical isomorphism $A^{**} \longrightarrow A$ in $\mathcal{K}$ is the identity.
(D) The canonical isomorphism $(A \otimes B)^* \longrightarrow A^* \otimes B^*$ in $\mathcal{K}$ is the identity.

Theorem 6. *If* J *satisfies* ***(I)*** *and* ***(D)****, then* $\mathrm{Mod}(Ax, J) \cong CCFC(Cl(Ax), J)$*.*

□

4 A concurrent λ-calculus and (de)compilation

In order to demonstrate the relevance of our semantic framework, this section tries to give a semantic reconstruction of fully-abstract compilation and decompilation from a higher-order calculus to the (first-order) π-calculus, such as [39,42]. We first design an instance of the computational λ-calculus [33], named

[6] Because we consider only the empty signature, the set of valuations is singleton.

$$\sigma ::= \tau \to \tau' \qquad \xi ::= \sigma \qquad \tau ::= (\xi_1, \ldots, \xi_n)$$
$$V ::= x \mid \lambda\langle\vec{x}\rangle.M$$
$$M ::= \langle\vec{V}\rangle \mid V\ \langle\vec{V}\rangle \mid \mathbf{let}\ \langle\vec{x}\rangle = M\ \mathbf{in}\ M'$$

(a) λ_c

$$\xi ::= \cdots \mid \sigma^*$$
$$V ::= \cdots \mid \mathbf{channel}_\sigma \mid \mathbf{send}_\sigma$$

(b) λ_{ch} (difference from λ_c)

Fig. 4. Syntax of types and terms of the λ_c- and λ_{ch}-calculi. The syntax of λ_c is adapted to the setting of this paper.

λ_{ch}, that is sound and complete with respect to compact closed Freyd categories. It is obtained by a straightforward extension of the coincidence between the computational λ-calculus and closed Freyd categories (Section 4.1). There are translations between π_F and λ_{ch} since both are sound and complete with respect to compact closed Freyd categories. Section 4.2 actually calculates the translations, and compare them with those in [39,42].

4.1 The λ_{ch}-calculus

The λ_{ch}-calculus is a computational λ-calculus with additional constructors dealing with channels. This section introduces and explains the calculus.

The situation is nicely expressed by the following intuitive equation:

$$\frac{\lambda_{ch}}{\lambda_c} \qquad \approx \qquad \frac{(\text{compact closed Freyd category} + \mathbf{I} + \mathbf{D})}{(\text{closed Freyd category})}.$$

The base calculus λ_c is the *computational λ-calculus*, which corresponds to closed Freyd category [33,37]. It is a call-by-value higher-order programming language, given in Fig. 4(a). Our calculus λ_{ch} is obtained by adding type and term constructors originating from the compact closed structure, which λ_c does not have.

Syntax As for types, λ_{ch} has a new constructor coming from the dual object A^*. Normalising occurrences of the dual A^* using the axioms **(I)** $A^{**} = A$ and **(D)** $(A \otimes B)^* = A^* \otimes B^*$, we obtain the following grammar of types:

$$\sigma ::= \tau \to \tau' \qquad \xi ::= \sigma \mid \sigma^* \qquad \tau ::= (\xi_1, \ldots, \xi_n)$$

where $n \geq 0$ and $(\xi_1, \ldots, \xi_n)$ is an alternative notation for $\xi_1 \otimes \cdots \otimes \xi_n$. Compared with λ_c, the only new type is the dual type σ^* of a function type σ.

As for terms, λ_{ch} has constructors corresponding to the unit and counit

$$\eta_A : I \longrightarrow A \otimes A^* \qquad \epsilon_A : A^* \otimes A \longrightarrow I \qquad \text{(for each object } A\text{)}$$

of the compact closed structure. We simply add these morphisms as constants:

$$\frac{}{\Gamma \vdash \mathbf{channel}_\sigma : () \to (\sigma, \sigma^*)} \qquad \text{and} \qquad \frac{}{\Gamma \vdash \mathbf{send}_\sigma : (\sigma^*, \sigma) \to ()}.$$

We shall often omit the subscript σ.

In summary, we obtain the syntax of λ_{ch} shown in Fig. 4. Interestingly, λ_{ch} can be seen as a very core of Concurrent ML [38], a practical higher-order concurrent language, although λ_{ch} is developed from purely semantic considerations.

Semantics Let us first discuss the intuitive meanings of the new constructors. The type σ^* is for *output channels*; $\mathbf{channel}\,\langle\rangle$ creates and returns a pair of an input channel and an output channel that are connected; and $\mathbf{send}\,\langle\alpha, V\rangle$ sends the value V via the output channel α. The following points are worth noting.

- λ_{ch} has no type constructor for *input channels*. The type system does not distinguish between input channels for type σ and values of type σ.
- λ_{ch} has no *receive* constructor. Receiving operation is implicit and on demand, delayed as much as possible.
- The send operator broadcasts a value via a channel. Several receivers may receive the same value from the same channel.

The first two points reflect the asynchrony of π_F, and the last point reflects the absence of non-replicated input (cf. Section 4.2).

Based on this intuition, we develop the operational, axiomatic and categorical semantics of λ_{ch}. We shall use the following abbreviations:

$$(\boldsymbol{\nu} xy)M \stackrel{\text{def}}{=} \mathbf{let}\,\langle x, y\rangle = \mathbf{channel}\,\langle\rangle\,\mathbf{in}\,M \qquad M \parallel N \stackrel{\text{def}}{=} \mathbf{let}\,\langle\rangle = M\,\mathbf{in}\,N.$$

Operational semantics Assume an infinite set $\mathcal{X}$ of *channels*, ranged over by α and β. For each channel α, we write α for the input name and $\bar{\alpha}$ for the output name, both of which are values. A *configuration* is a tuple $(M, \vec{\alpha}, \mu)$ of a term M, a sequence $\vec{\alpha}$ of generated channels and a sequence μ of performed send operations, i.e. $\mu = (\mathbf{send}\,\langle\bar{\beta}_1, V_1\rangle, \ldots, \mathbf{send}\,\langle\bar{\beta}_k, V_k\rangle)$. The *reduction relation* is defined by the following rules for channels

$$\begin{aligned}
(E[\mathbf{channel}\,\langle\rangle],\ \vec{\alpha},\ \mu) &\longrightarrow (E[\langle\beta, \bar{\beta}\rangle],\ \vec{\alpha}\cdot\beta,\ \mu) && (\beta \notin \vec{\alpha})\\
(E[\mathbf{send}\,\langle\bar{\beta}, V\rangle],\ \vec{\alpha},\ \mu) &\longrightarrow (E[\langle\rangle],\ \vec{\alpha},\ \mu\cdot\mathbf{send}\,\langle\bar{\beta}, V\rangle)\\
(E[\beta\, V],\ \vec{\alpha},\ \mu) &\longrightarrow (E[W\, V],\ \vec{\alpha},\ \mu) && (\mathbf{send}\,\langle\bar{\beta}, W\rangle \in \mu).
\end{aligned}$$

in addition to the standard rules for λ-abstractions and let-expressions, which change only M. Here the set of *evaluation contexts* is given by the grammar:

$$E ::= [] \mid \mathbf{let}\,\langle\vec{x}\rangle = E\,\mathbf{in}\,M \mid \mathbf{let}\,\langle\vec{x}\rangle = M\,\mathbf{in}\,E.$$

Note that M and N in $\mathbf{let}\,\langle\vec{x}\rangle = M\,\mathbf{in}\,N$ are evaluated in parallel (cf. Remark 3). This justifies the notation $M \parallel N$, an abbreviation for $\mathbf{let}\,\langle\rangle = M\,\mathbf{in}\,N$.

Axiomatic semantics The inference rules of the equational logic for λ_{ch} are those for λ_c with the rule of concurrent evaluation

$$\mathbf{let}\,\langle\vec{x}\rangle = M\,\mathbf{in}\,\mathbf{let}\,\langle\vec{y}\rangle = N\,\mathbf{in}\,L \quad = \quad \mathbf{let}\,\langle\vec{y}\rangle = N\,\mathbf{in}\,\mathbf{let}\,\langle\vec{x}\rangle = M\,\mathbf{in}\,L \qquad ;$$

the β- and η-rules for channels

$$\begin{array}{rcl}(\boldsymbol{\nu} x\bar{x})(\mathbf{send}\,\langle \bar{x}, V\rangle \parallel M) & = & (\boldsymbol{\nu} x\bar{x})(\mathbf{send}\,\langle \bar{x}, V\rangle \parallel M\{V/x\}) \\ (\boldsymbol{\nu} y\bar{y})(\mathbf{send}\,\langle \bar{z}, y\rangle \parallel N) & = & N\{\bar{z}/\bar{y}\}\end{array}$$

where $\bar{x} \notin \mathbf{Fv}(V) \cup \mathbf{Fv}(M)$, $y \notin \mathbf{Fv}(N)$ and $\bar{z} \neq \bar{y}$; and a GC rule.

Categorical semantics One can interpret λ_{ch}-terms in a compact closed Freyd category with **(I)** and **(D)**. The interpretation of the λ_c-calculus part is standard [37,24]; the constant $\mathbf{channel}_\sigma$ (resp. $\mathbf{send}_\sigma$) is interpreted as the "closure" whose body is η_σ (resp. ϵ_σ) as expected.

$$[\![\Gamma \vdash \mathbf{channel}_\sigma : () \to (\sigma, \sigma^*)]\!] \stackrel{\text{def}}{=} J(!_\Gamma; \Lambda_{I,I,\sigma\otimes\sigma^*}(\eta_\sigma))$$
$$[\![\Gamma \vdash \mathbf{send}_\sigma : (\sigma^*, \sigma) \to ()]\!] \stackrel{\text{def}}{=} J(!_\Gamma; \Lambda_{I,\sigma\otimes\sigma^*,I}(\epsilon_\sigma)).$$

The categorical semantics is sound and complete with respect to the equational theory of the λ_{ch}-calculus. The proofs are basically straightforward but there is a subtle issue in the definition of the term model: we have different definitions of the right adjoint $I \Rightarrow (-)$, which are of course equivalent but do not coincide on the nose. Our choice here is $I \Rightarrow \langle\vec{\xi}\rangle \stackrel{\text{def}}{=} (\vec{\xi}^{\perp}) \to ()$.

4.2 Translations between $\boldsymbol{\lambda_{ch}}$ and $\boldsymbol{\pi_F}$

The higher-order calculus λ_{ch} is equivalent to π_F. This is because both calculi correspond to the same class of categories, namely, the class of compact closed Freyd categories with **(I)** and **(D)**, i.e.,

$$(\lambda_{ch}) \;\approx\; (\text{compact closed Freyd category} + \mathbf{I} + \mathbf{D}) \;\approx\; (\pi_F).$$

This subsection studies translations derived from this semantic correspondence.

The translations are defined by the interpretations in the term models. For example, the translation $(\!|-|\!)$ from λ_{ch} to π_F is induced by the interpretation of λ_{ch}-terms in the term model $Cl(\emptyset)$. The interpretation $[\![M]\!]_{Cl(\emptyset)}$ of a λ_{ch}-term M is an equivalence class of π_F-processes, since a morphism in $Cl(\emptyset)$ is an equivalence class of π_F-processes. The translation $(\!|M|\!)$ is defined by choosing a representative of the equivalence class. The other direction $[\![-]\!]$ is obtained by the interpretation of π_F in the term model of λ_{ch}.

Figures 5 and 6 are concrete definitions of the translations for a natural choice of representatives. Let us discuss the translations in more details.

The translation from π_F to λ_{ch} (Fig. 5) is easy to understand. It directly expresses the higher-order view of the first-order π-calculus. For example, an output action is mapped to an application and an input-prefixing $!a(\vec{x}).P$ to a send operation of the value $\lambda\langle\vec{x}\rangle.P$ via the channel a.

An interesting (and perhaps confusing) phenomenon is that an input channel in π_F is mapped to an output channel in λ_{ch}. This can be explained as follows. In the name-passing viewpoint, the reduction

$$(\boldsymbol{\nu} xy)(!y(\vec{z}).P \mid x\langle\vec{u}\rangle) \quad\longrightarrow\quad (\boldsymbol{\nu} xy)(!y(\vec{z}).P \mid P\{\vec{u}/\vec{z}\})$$

$$\llbracket \mathbf{ch}^o[\vec{T}] \rrbracket \stackrel{\text{def}}{=} \llbracket \vec{T} \rrbracket \to () \qquad \llbracket \mathbf{ch}^i[\vec{T}] \rrbracket \stackrel{\text{def}}{=} (\llbracket \vec{T} \rrbracket \to ())^* \qquad \llbracket (T_1, \ldots, T_n) \rrbracket \stackrel{\text{def}}{=} (\llbracket T_1 \rrbracket, \ldots, \llbracket T_n \rrbracket)$$

$$\llbracket \mathbf{0} \rrbracket \stackrel{\text{def}}{=} \langle\rangle \qquad \llbracket P \mid Q \rrbracket \stackrel{\text{def}}{=} \llbracket P \rrbracket \parallel \llbracket Q \rrbracket \qquad \llbracket (\boldsymbol{\nu} xy) P \rrbracket \stackrel{\text{def}}{=} (\boldsymbol{\nu} xy) \llbracket P \rrbracket$$

$$\llbracket \bar{a}\langle \vec{x} \rangle \rrbracket \stackrel{\text{def}}{=} \bar{a}\,\langle \vec{x} \rangle \qquad \llbracket !a(\vec{x}).P \rrbracket \stackrel{\text{def}}{=} \mathbf{send}\,\langle a, \lambda(\vec{x}).\llbracket P \rrbracket \rangle$$

Fig. 5. Translation from π_F to λ_{ch}

$$(\!|\tau_1 \to \tau_2|\!) \stackrel{\text{def}}{=} \mathbf{ch}^o[(\!|\tau_1|\!), (\!|\tau_2|\!)^\perp] \qquad (\!|\sigma^*|\!) \stackrel{\text{def}}{=} (\!|\sigma|\!)^\perp \qquad (\!|(\tau_1, \ldots, \tau_n)|\!) \stackrel{\text{def}}{=} ((\!|\tau_1|\!), \ldots, (\!|\tau_n|\!))$$

$$(\!|x|\!)_p \stackrel{\text{def}}{=} (p \leftrightharpoons x) \qquad (\!|\lambda \vec{x}.M|\!)_p \stackrel{\text{def}}{=} !p(\vec{x}, \vec{q}).(\!|M|\!)_{\vec{q}} \qquad (\!|\langle \vec{V} \rangle|\!)_{\vec{p}} \stackrel{\text{def}}{=} (\!|V_1|\!)_{p_1} \mid \cdots \mid (\!|V_n|\!)_{p_n}$$

$$(\!|V\,\langle \vec{W} \rangle|\!)_{\vec{p}} \stackrel{\text{def}}{=} (\boldsymbol{\nu} a\bar{a})(\boldsymbol{\nu} \vec{r}\vec{s})((\!|V|\!)_a \mid (\!|\langle \vec{W} \rangle|\!)_{\vec{s}} \mid \bar{a}\langle \vec{r}, \vec{p} \rangle)$$

$$(\!|\mathbf{let}\,\langle \vec{x} \rangle = M\,\mathbf{in}\,N|\!)_{\vec{p}} \stackrel{\text{def}}{=} (\boldsymbol{\nu} \vec{x}\vec{q})((\!|M|\!)_{\vec{q}} \mid (\!|N|\!)_{\vec{p}})$$

$$(\!|\mathbf{channel}|\!)_p \stackrel{\text{def}}{=} !p(x, y).x \hookrightarrow y \qquad (\!|\mathbf{send}|\!)_p \stackrel{\text{def}}{=} !p(x, y).x \hookrightarrow y$$

Fig. 6. Translation from λ_{ch} to π_F

sends $\vec{u}$ to the process $!y(\vec{z}).P$, and thus x is output and y is input. In the process-passing viewpoint, the abstraction $(\vec{z}).P$ is sent to the location of x, and thus y is the output and x is the input.

Next, we explain the translation from λ_{ch} to π_F (Fig. 6).

Let us first examine the translation of types. The most non-trivial part is the translation of a function type $\tau_1 \to \tau_2$. A key to understand the translation is the isomorphism $\tau_1 \to \tau_2 \cong \tau_1 \otimes \tau_2^\perp \to ()$. The latter form of function type corresponds to an output channel type in π_F. Hence a function is understood as a process additionally taking channels to which the return values are passed.

The translation $(\!|M|\!)_{\vec{p}}$ of a λ_{ch}-term $\Gamma \vdash M : (\xi_1, \ldots, \xi_n)$ takes extra parameters $\vec{p} = p_1, \ldots, p_n$ to which the values should be placed. This is a consequence of the definition in the π_F-term model that a morphism $\vec{T} \longrightarrow \vec{S}$ is a process $\vec{x}\colon \vec{T}$, $\vec{y}\colon \vec{S}^\perp \vdash P : \diamond$. Here $\vec{p}$ corresponds to $\vec{y}$, Γ to $\vec{x}\colon \vec{T}$ and $\vec{\xi}$ to $\vec{S}$.

Now it is not so difficult to understand the interpretations of constructs in the λ_c-calculus. For example, the abstraction $(\!|\lambda\langle \vec{x} \rangle.M|\!)_p$ is mapped to an abstraction $(\vec{x}, \vec{q}).(\!|M|\!)_{\vec{q}}$ placed at p, which takes additional channels $\vec{q}$ to which the results of the evaluation of M should be sent.

It might be surprising that the interpretations of **channel** and **send** coincide. This is because of the one-sided formulation of π_F. In the two-sided formulation, the unit η and counit ϵ of the compact closed structure, corresponding to **channel** and **send**, can be written as logical inference rules

$$\frac{\Gamma, A, A^\perp \vdash \Delta}{\Gamma \vdash \Delta} \qquad \text{and} \qquad \frac{\Gamma \vdash A^\perp, A, \Delta}{\Gamma \vdash \Delta},$$

which are different. In the one-sided formulation, however, they become

$$\frac{\Gamma, A, A^\perp, \Delta^\perp \vdash}{\Gamma, \Delta^\perp \vdash}.$$

$$(\!|\mathbf{0}|\!) \stackrel{\text{def}}{=} \mathbf{0} \qquad (\!|P \mid Q|\!) \stackrel{\text{def}}{=} (\!|P|\!) \mid (\!|Q|\!) \qquad (\!|(\boldsymbol{\nu} xy)P|\!) \stackrel{\text{def}}{=} (\boldsymbol{\nu} xy)(\!|P|\!) \qquad (\!|\,!x\,v\,|\!) \stackrel{\text{def}}{=} (\!|v|\!)_x$$

$$(\!|v\langle w_1, \ldots, w_n\rangle|\!) \stackrel{\text{def}}{=} (\boldsymbol{\nu}\bar{a}a)(\boldsymbol{\nu}\bar{b}_1 b_1)\ldots(\boldsymbol{\nu}\bar{b}_n b_n)((\!|v|\!)_a \mid (\!|w_1|\!)_{b_1} \mid \cdots \mid (\!|w_n|\!)_{b_n} \mid \bar{a}\langle\bar{b}_1, \ldots, \bar{b}_n\rangle)$$

$$(\!|x|\!)_a \stackrel{\text{def}}{=} (a \hookrightarrow x) \qquad (\!|(\vec{x}).P|\!)_a \stackrel{\text{def}}{=} !a(\vec{x}).(\!|P|\!)$$

Fig. 7. Translation from AHOπ to π_F

Hence η and ϵ (or **channel** and **send**) cannot be distinguished in π_F.

The translation $(\!|-|\!)$ must be the inverse of $[\![-]\!]$ because both the term models are the initial compact closed Freyd category with **(I)** and **(D)**. That means, $\emptyset \rhd \Gamma \vdash P = (\!|[\![P]\!]|\!)$ and $\emptyset \rhd \Gamma \vdash M = [\![(\!|M|\!)]\!]$ are provable for every P and M. This result is independent of the choice of representatives.

4.3 Relation to other calculi and translations

A number of higher-order concurrent calculi, as well as their translations to the first-order π-calculus, have been proposed and studied (e.g. [29,47,39,40,42,45]). The calculus λ_{ch} and the translations have a lot of ideas in common with those calculi and translations; see Section 6.

This subsection mainly discusses the relationship to the translations by Sangiorgi [42] (see also [43]) between *asynchronous higher-order π-calculus* (*AHOπ* for short) and *asynchronous local π-calculus* (*Lπ* for short). Here we focus on this work because it is closest to ours. We shall see that our semantic or categorical development provides us with a semantic reconstruction of Sangiorgi's translations, as well as an extension.

A variant of AHOπ can be seen as a fragment of λ_{ch}. The syntax of processes of AHOπ and representation by λ_{ch}-terms are given as follow:

$$\begin{array}{rrrrrrrrrrr} v, w & ::= & x & \mid & (\vec{x}).P & \qquad P, Q & ::= & \mathbf{0} & \mid & (P \mid Q) & \mid & (\boldsymbol{\nu} xy)P & \mid & !x\,v & \mid & v\langle\vec{w}\rangle \\ & & x & & \lambda\langle\vec{x}\rangle.P & & & \langle\rangle & & P \parallel Q & & (\boldsymbol{\nu} xy)P & & \mathbf{send}\,\langle x, v\rangle & & v\,\langle\vec{w}\rangle. \end{array}$$

(It slightly differs from the original syntax, as $\boldsymbol{\nu}$ binds a pair of names.)

This fragment is nicely described as the limitation on types:

$$\sigma ::= (\vec{\sigma}) \to () \qquad \xi ::= \sigma \mid \sigma^* \qquad \tau ::= ().$$

Recall that σ is a type for abstractions, ξ is a type for variables, and τ is a type for terms. This limitation means that (1) an abstraction cannot take a channel as an argument, and (2) a term M must be of the unit type, i.e. a process.

Once regarding AHOπ as a fragment of λ_{ch}, the translation from AHOπ to π_F is obtained by restricting $(\!|-|\!)$ to AHOπ. The resulting translation is in Fig. 7. As mentioned, the translation is the same as that of Sangiorgi [42] except for minor differences due to the slight change of the syntax.

Sangiorgi also gave a translation in the opposite direction, from Lπ to AHOπ in the same paper. The calculus Lπ is a fragment of the π-calculus in which only

output channels can be passed. The **i**/**o**-separation of π_F allows us to characterise the local version of π_F by a limitation on types. In the local variant, the output channel type is restricted to $T ::= \mathbf{ch}^o[\vec{T}]$, expressing that only output channels can be passed via an output channel. Then the definition of type environment should be changed accordingly: $\Gamma ::= \cdot \mid x : T \mid x : T^\perp$ (since the syntactic class represented by T is not closed under the dual $(-)^\perp$ in the local setting).

Interestingly the limitation on types in AHOπ coincides with that in Lπ, when one identify $\mathbf{ch}^o[\vec{T}]$ with $(\vec{T}) \to ()$ (as we have done in many places). In other words, the syntactic restrictions of AHOπ and Lπ are the same semantic conditions described in different syntax. As a consequence, the image of Lπ by $[\![-]\!]$ is indeed in AHOπ.

Remark 4. There is, however, a notable difference from Sangiorgi's work [42]. Sangiorgi proved that the translation is fully-abstract with respect to barbed congruence; in contrast, we only show that $\vdash M = N$ iff $\vdash (\!|M|\!) = (\!|N|\!)$. In particular, the η-rule is inevitable for our argument. The presence of the η-rules significantly simplifies the argument, at the cost of operational justification (recall that the η-rule is not sound with respect to barbed congruence).

It is natural to ask how one can reconstruct the full-abstraction result with respect to barbed congruence. An interesting observation is that, if M and N are AHOπ processes, then $\vdash^\ominus M = N$ iff $\vdash^\ominus (\!|M|\!) = (\!|N|\!)$, where $\vdash^\ominus$ means provability without using η-rules. We expect that this semantic observation explains why locality is essential as noted in [42]; we leave the details for future work. □

5 Discussions

Connection to logics We have so far studied a connection between compact closed Freyd category and π-calculus. Here we briefly discuss the missing piece of the Curry-Howard-Lambek correspondence, namely logic.

The model of this paper is closely related to linear logic. Actually, every compact closed Freyd category is a model of linear logic (more precisely, MELL), as an instance of linear-non-linear model [6] (see, e.g., [27] for categorical models of linear logic). The interpretation of formulas is shown in Table 1. It differs from the translations by Abramsky [1] and Bellin and Scott [5] and from the Curry-Howard correspondence for session types by Caires and Pfenning [8], but resembles the connection between a variant of local π-calculus and a polarised linear logic by Honda and Laurent [19]; a detailed analysis of the translation is left for future work.

The logic corresponding to compact closed Freyd category should be a proper extension of linear logic, since compact closed Freyd categories form a proper subclass of linear-non-linear models. For example, the following rules are invalid in linear logic but admissible in compact closed Freyd categories:

$$\frac{\vdash \Gamma \qquad \vdash \Delta}{\vdash \Gamma, \Delta} \qquad \frac{\vdash \Gamma, A, B \qquad \vdash \Delta, A^\perp, B^\perp}{\vdash \Gamma, \Delta} \qquad \frac{\vdash \Gamma, A, A^\perp}{\vdash \Gamma}.$$

linear logic (formula)	compact closed Freyd category (object)	π_F-calculus (type environment)
$A \otimes B$ $A \parr B$	$A \otimes B$	$x : A, y : B$
$!A$	$I \Rightarrow A$	$x : \mathbf{ch}^o[A^\perp]$
$?A$	$(A \Rightarrow I)^*$	$x : \mathbf{ch}^i[A]$

Table 1. The categorical and π_F-calculus interpretations of MELL formulas

These rules, especially the second rule called *multicut*, were often studied in concurrency theory; see Abramsky et al. [2] for their relevance to concurrency.

Do the above rules fill the gap between linear logic and compact closed Freyd category? Recent work by Hasegawa [15] suggests that MELL with above rules is still weaker than compact closed Freyd category. First observe that the above rules can be interpreted in any linear-non-linear model of which the monoidal category is compact closed. Hasegawa showed that a linear-non-linear model whose monoidal category is compact closed induces a closed Freyd category of which the monoidal category is *traced* (and vice versa) but the induced Freyd category is not necessarily compact closed. Hence the logic corresponding to compact closed Freyd category has further axioms or rules in addition to the above ones. A reasonable candidate for the additional axiom is $! \cong ?$; interestingly, Atkey et al. [3] reached a similar rule from a different perspective. Further investigation is left for future work.

Non-empty signature The categorical type theory for the λ-calculus considers a family parameterised by *signatures*, consisting of atomic types and constants. It covers, for example, the λ-calculus with natural number type and arithmetic constants (such as addition and multiplication), as well as a calculus with integer reference type and read and update functions.

Although this paper only considers the calculus with the empty signature, which has no additional type nor constant, extending our theory to handle non-empty signatures is, in a sense, not difficult. The easiest way is to apply the established theory of the computational λ-calculus [33,37]. As we have seen in Section 4, the π_F-calculus can be seen as a computational λ-calculus λ_{ch} having constants for manipulating channels; hence the π_F-calculus with additional constants is λ_{ch} with the additional constants, which is still in the family of computational λ-calculus.

The π_F-calculus with non-empty signature has several applications. We shall briefly discuss some of them.

An important example of π_F with non-empty signature is the calculus with non-replicated input, which we regard as a calculus with additional "process constants" but without any additional type. A key observation is that every non-replicated input process $a(\vec{x}).P$ can be expressed as

$$a(\vec{x}).P \;\cong^c\; (\boldsymbol{\nu}\bar{b}b)(a(\vec{x}).\bar{b}\langle\vec{x}\rangle \mid !b(\vec{x}).P) \qquad (\cong^c \text{ is weak barbed congruence})$$

and thus it suffices to deal with non-replicated input processes in special form, namely $a : \mathbf{ch}^i[\vec{T}],\ \bar{b} : \mathbf{ch}^o[\vec{T}] \vdash a(\vec{x}).\bar{b}\langle\vec{x}\rangle : \diamond$. Adding these processes as con-

stants and the computational rules of $a(\vec{x}).\bar{b}\langle\vec{x}\rangle$ as equational axioms results in a calculus with non-replicated inputs. The categorical model is a compact closed Freyd category with distinguished morphisms $(A \Rightarrow I) \longrightarrow (A \Rightarrow I)$ for each object A which satisfy certain axioms.

This technique is applicable to synchronous output as well. Because

$$\bar{a}\langle\vec{x}\rangle.P \;\cong^c\; (\boldsymbol{\nu}\bar{b}b)(\bar{a}\langle\vec{x}\rangle.\bar{b}\langle\rangle \mid !b().P),$$

it suffices to consider constants representing $\bar{a}\colon\mathbf{ch}^o[\vec{T}], \vec{x}\colon\vec{T}, \bar{b}\colon\mathbf{ch}^o[] \vdash \bar{a}\langle\vec{x}\rangle.\bar{b}\langle\rangle : \diamond$.

6 Related Work

Logical studies of $\boldsymbol{\pi}$-calculi There is a considerable amount of studies on connections between process calculi and linear logic. Here we divide these studies into two classes. These classes are substantially different; for example, one regards the formula $A \otimes B$ as a type for processes with two "ports" of type A and B, whereas the other as the session-type $!A.B$. Our work is more closely related to the former than the latter, but some interesting coincidence to the latter kind of studies can also be found.

The former class of research dates back to the work by Abramsky [1] and Bellin and Scott [5], where they discovered that π-calculus processes can encode proof-nets of classical linear logic. Later, Abramsky et al. [2] introduced the *interaction categories* to give a semantic description of a CCS-like process calculus. In their work, they observed that the compact closed structure is important to capture the strong expressive power of process calculi.

A tighter connection between π-calculus and proof-nets was recently presented by Honda and Laurent [19]. They showed that an **i**/**o**-typed π-calculus corresponds to *polarised proof-nets*, and introduced the notion of *extended reduction* for the π-calculus to simulate cut-elimination. The π-calculus used in this work is very similar to π_F in terms of syntax and reduction. Their calculus is asynchronous, does not allow non-replicated inputs, and requires **i**/**o**-separation. Furthermore, the extended reduction is almost the same as the rules (E-Beta) and (E-GC) except for the side conditions. A significant difference compared to our work is that their calculus is *local* [28,49], reflecting the fact that the corresponding logic is polarised.

Our work is inspired by these studies. The idea of **i**/**o**-separation can already be found in the work by Bellin and Scott and the use of compact closed category is motivated by the study of interaction category. It is worth mentioning here that the design of π_F is also influenced by the calculus introduced by Laird [22], although it is not a logical study but categorical (see below).

The latter approach started with the Curry-Howard correspondences between session-typed π-calculi and linear logic established by Caires, Pfenning and Toninho [8,9] and subsequently by Wadler [48]. These correspondences are exact in the sense that every process has a corresponding proof, and vice versa. As a consequence, processes of the calculi inherit good properties of linear logic proofs

such as termination and confluence of cut-elimination. In terms of process calculi, process of these calculi do not fall into deadlock or race condition. This can be seen as a serious restriction of expressive power [48,3,26].

Several extensions to increase the expressiveness of these calculi have been proposed and studied. Interestingly, ideas behind some of these extensions are related to our work, in particular to Section 5 discussing the multicut rule [2] and the axiom $! \cong ?$. Atkey et al. [3] studied *CP* [48] with the multicut rule and $! \cong ?$ and discussed how these extensions increase the expressiveness of the calculus, at the cost of losing some good properties of CP. Dardha and Gay [10] studied another extension of CP with multicut, keeping the calculus deadlock-free by an elaborated type system.

Balzer and Pfenning [4] proposed a session-typed calculus with shared (mutable) resources, inspired by linear-non-linear adjunction [6].

Categorical semantics of π-calculi The idea of using a closed Freyd category to model the π-calculus is strongly inspired by Laird [22]. He introduced the *distributive-closed Freyd category* to describe abstract properties of a game-semantic model of the asynchronous π-calculus and showed that distributive-closed Freyd categories with some additional structures suffice to interpret the asynchronous π-calculus. The additional structures are specific to his game model and not completely axiomatised.[7] Our notion of compact closed Freyd category might be seen as a reformulation of his idea, obtained by filtering out some structures difficult to axiomatise and by strengthening some others to make axioms simpler. A significant difference is that our categorical model does not deal with non-replicated inputs, which we think is essential for a simple axiomatisation.

Another approach for categorical semantics of the π-calculus has been the presheaf based approach [44,12]. These studies gave particular categories that nicely handles the nominal aspects of the π-calculus; these studies, however, do not aim for a correspondence between a categorical structure and the π-calculus.

Higher-order calculi with channels Besides the λ_{ch}-calculus, there are numbers of functional languages augmented by communication channels, from theoretical ones [13,46,48,25] to practical languages [38,34].

On the practical side, Concurrent ML (CML) [38], among others, is a well-developed higher-order concurrent language. CML has primitives to create channels and threads, and primitives to send and accept values through channels. Since our λ_{ch}-calculus can create (non-linear) channels and send values via channels, the λ_{ch}-calculus can be seen as a core calculus of CML despite its origin in categorical semantics. The major difference between CML and the λ_{ch}-calculus is that communications in CML are synchronous whereas communications in the λ_{ch}-calculus are asynchronous.

On the theoretical side, session-typed functional languages have been actively studied [13,46,48,25]. Notably, some of these languages [46,48,25] are built upon

[7] A list of properties in [22] does not seem to be complete. We could not prove some claims in the paper only from these properties, but with ones specific to his model.

the Curry-Howard foundation between linear logic and session-typed processes. It might be interesting to investigate whether we can relate these languages and the λ_{ch}-calculus through the lens of Curry-Howard-Lambek correspondence.

Higher-order vs. first-order π-calculus A number of translations from higher-order languages to the π-calculus have been developed [47,39,40,42,45] since Milner [29] presented the encodings of the λ-calculus into the π-calculus. The basic idea shared by these studies is to transform $\lambda x.M$ to a process $!a(x,p).P$ that receives the argument x together with a name p where the rest of the computation will be transmitted. In our framework, this idea is described as the isomorphism $A \Rightarrow B \cong A \otimes B^* \Rightarrow I$.

Among others, the translation from AHOπ to Lπ [42] is the closest to our translation from the λ_{ch}-calculus to the π_F-calculus. Sangiorgi [41] observed that Milner's translation can be established via the translation of AHOπ by applying the CPS transformation to the λ-calculus. This observation also applies to our translation. That is, we can obtain Milner's translation by combining CPS transformation and the compilation of the λ_{ch}-calculus.

7 Conclusion and future work

We have introduced an **i**/**o**-typed π-calculus (π_F-calculus) as well as the categorical counterpart of π_F-calculus (compact closed Freyd category) and showed the categorical type theory correspondence between them. The correspondence was established by regarding the π-calculus as a higher-order programming language, introducing the **i**/**o**-separation, and introducing the η-rule, a rule that explains the mismatch between behavioural equivalences and categorical models.

As an application of our semantic framework we introduced a higher-order calculus λ_{ch}-calculus "equivalent" to the π_F-calculus. We have demonstrated that translations between λ_{ch}-calculus and π_F-calculus can be derived by a simple semantic argument, and showed that the translation from λ_{ch} to π_F is a generalisation of the translation from AHOπ to Lπ given by Sangiorgi [42].

There are three main directions for future work. First, further investigation on the η-rule is indispensable. We plan to construct a categorical model of the π_F-calculus with an additional constant that captures barbed congruence. Revealing the relationship between locality and the η-rule is another important problem. Second, the operational properties of the λ_{ch}-calculus and its relation to the equational theory needs a further investigation. Third, finding the logical counterpart of compact closed Freyd category to establish a proper Curry-Howard-Lambek correspondence is an interesting future work.

Acknowledgement.

We would like to thank Naoki Kobayashi, Masahito Hasegawa and James Laird for discussions, and anonymous referees for valuable comments. This work was supported by JSPS KAKENHI Grant Number 15H05706 and 16K16004.

References

1. Abramsky, S.: Proofs as processes. Theor. Comput. Sci. **135**(1), 5–9 (1994)
2. Abramsky, S., Gay, S.J., Nagarajan, R.: Interaction categories and the foundations of typed concurrent programming. In: Proceedings of the NATO Advanced Study Institute on Deductive Program Design, Marktoberdorf, Germany. pp. 35–113 (1996)
3. Atkey, R., Lindley, S., Morris, J.G.: Conflation confers concurrency. In: A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday. pp. 32–55 (2016)
4. Balzer, S., Pfenning, F.: Manifest sharing with session types. PACMPL **1**(ICFP), 37:1–37:29 (2017)
5. Bellin, G., Scott, P.J.: On the π-calculus and linear logic. Theor. Comput. Sci. **135**(1), 11–65 (1994)
6. Benton, P.N.: A mixed linear and non-linear logic: Proofs, terms and models. In: Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers. pp. 121–135 (1995)
7. Boreale, M.: On the expressiveness of internal mobility in name-passing calculi. Theor. Comput. Sci. **195**(2), 205–226 (1998)
8. Caires, L., Pfenning, F.: Session types as intuitionistic linear propositions. In: CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings. pp. 222–236 (2010)
9. Caires, L., Pfenning, F., Toninho, B.: Linear logic propositions as session types. Mathematical Structures in Computer Science **26**(3), 367–423 (2016)
10. Dardha, O., Gay, S.J.: A new linear logic for deadlock-free session-typed processes. In: Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings. pp. 91–109 (2018)
11. De Nicola, R., Hennessy, M.: Testing equivalence for processes. In: Automata, Languages and Programming, 10th Colloquium, Barcelona, Spain, July 18-22, 1983, Proceedings. pp. 548–560 (1983)
12. Fiore, M.P., Moggi, E., Sangiorgi, D.: A fully abstract model for the π-calculus. Inf. Comput. **179**(1), 76–117 (2002)
13. Gay, S.J., Vasconcelos, V.T.: Linear type theory for asynchronous session types. J. Funct. Program. **20**(1), 19–50 (2010)
14. Girard, J.: Linear logic. Theor. Comput. Sci. **50**, 1–102 (1987)
15. Hasegawa, M.: From linear logic to cyclic sharing. Lecture slides, Linearity (2018)
16. Hayashi, S.: Adjunction of semifunctors: Categorical structures in nonextensional lambda calculus. Theor. Comput. Sci. **41**, 95–104 (1985)
17. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)
18. Honda, K.: Types for dyadic interaction. In: CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings. pp. 509–523 (1993)
19. Honda, K., Laurent, O.: An exact correspondence between a typed pi-calculus and polarised proof-nets. Theor. Comput. Sci. **411**(22-24), 2223–2238 (2010)
20. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Programming Languages and Systems - ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software,

ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings. pp. 122–138 (1998)
21. Kelly, G.M., Laplaza, M.L.: Coherence for compact closed categories. Journal of Pure and Applied Algebra **19**, 193–213 (1980)
22. Laird, J.: A game semantics of the asynchronous π-calculus. In: CONCUR 2005 - Concurrency Theory, 16th International Conference. pp. 51–65 (2005)
23. Lambek, J., Scott, P.J.: Introduction to higher-order categorical logic, vol. 7. Cambridge University Press (1988)
24. Levy, P.B., Power, J., Thielecke, H.: Modelling environments in call-by-value programming languages. Inf. Comput. **185**(2), 182–210 (2003)
25. Lindley, S., Morris, J.G.: A semantics for propositions as sessions. In: Programming Languages and Systems - 24th European Symposium on Programming, ESOP 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings. pp. 560–584 (2015)
26. Mazza, D.: The true concurrency of differential interaction nets. Mathematical Structures in Computer Science **28**(7), 1097–1125 (2018)
27. Melliès, P.A.: Categorical semantics of linear logic. Panoramas et syntheses **27**, 15–215 (2009)
28. Merro, M.: Locality in the π-calculus and applications to distributed objects. Ph.D. thesis, École Nationale Supérieure des Mines de Paris (2000)
29. Milner, R.: Functions as processes. Mathematical Structures in Computer Science **2**(2), 119–141 (1992)
30. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, I. Inf. Comput. **100**(1), 1–40 (1992)
31. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, II. Inf. Comput. **100**(1), 41–77 (1992)
32. Milner, R., Sangiorgi, D.: Barbed bisimulation. In: Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings. pp. 685–695 (1992)
33. Moggi, E.: Computational lambda-calculus and monads. In: Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989. pp. 14–23 (1989)
34. Peyton Jones, S.L., Gordon, A.D., Finne, S.: Concurrent Haskell. In: Conference Record of POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, St. Petersburg Beach, Florida, USA, January 21-24, 1996. pp. 295–308 (1996)
35. Pierce, B.C., Sangiorgi, D.: Typing and subtyping for mobile processes. Mathematical Structures in Computer Science **6**(5), 409–453 (1996)
36. Power, J., Robinson, E.: Premonoidal categories and notions of computation. Mathematical Structures in Computer Science **7**(5), 453–468 (1997)
37. Power, J., Thielecke, H.: Closed Freyd- and kappa-categories. In: Automata, Languages and Programming, 26th International Colloquium, ICALP'99. pp. 625–634 (1999)
38. Reppy, J.H.: CML: A higher-order concurrent language. In: Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation (PLDI), Toronto, Ontario, Canada, June 26-28, 1991. pp. 293–305 (1991)
39. Sangiorgi, D.: Expressing mobility in process algebras : first-order and higher-order paradigms. Ph.D. thesis, University of Edinburgh, UK (1993)
40. Sangiorgi, D.: π-Calculus, internal mobility, and agent-passing calculi. Theor. Comput. Sci. **167**(1&2), 235–274 (1996)

41. Sangiorgi, D.: From λ to π; or, Rediscovering continuations. Mathematical Structures in Computer Science **9**(4), 367–401 (1999)
42. Sangiorgi, D.: Asynchronous process calculi: the first- and higher-order paradigms. Theor. Comput. Sci. **253**(2), 311–350 (2001)
43. Sangiorgi, D., Walker, D.: The π-calculus—A Theory of Mobile Processes. Cambridge University Press (2001)
44. Stark, I.: 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. pp. 36–42 (1996)
45. Toninho, B., Caires, L., Pfenning, F.: Functions as session-typed processes. In: Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings. pp. 346–360 (2012)
46. Toninho, B., Caires, L., Pfenning, F.: Higher-order processes, functions, and sessions: A monadic integration. In: Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings. pp. 350–369 (2013)
47. Turner, D.N.: The polymorphic Pi-calculus : theory and implementation. Ph.D. thesis, University of Edinburgh, UK (1996)
48. Wadler, P.: Propositions as sessions. J. Funct. Program. **24**(2-3), 384–418 (2014)
49. Yoshida, N.: Minimality and separation results on asynchronous mobile processes – representability theorems by concurrent combinators. Theor. Comput. Sci. **274**(1-2), 231–276 (2002)