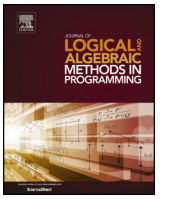
Journal of Logical and Algebraic Methods in Programming 124 (2022) 100717

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp



Comparing type systems for deadlock freedom

Ornela Dardha [a,*], Jorge A. Pérez [b,*]

[a] School of Computing Science, University of Glasgow, UK
[b] University of Groningen, the Netherlands

ARTICLE INFO

Article history:
Received 30 October 2020
Received in revised form 24 August 2021
Accepted 1 September 2021
Available online 10 September 2021

Keywords:
Concurrency
Process calculi
Linear types
Session types
Deadlock freedom
Linear logic

ABSTRACT

Message-passing software systems exhibit non-trivial forms of concurrency and distribution; they are expected to follow intended protocols among communicating services, but also to never "get stuck". This intuitive requirement has been expressed by liveness properties such as progress or (dead)lock freedom and various type systems ensure these properties for concurrent processes. Unfortunately, very little is known about the precise relationship between these type systems and the classes of typed processes they induce.
This paper puts forward the first comparative study of different type systems for message-passing processes that guarantee deadlock freedom. We compare two classes of deadlock-free typed processes, here denoted $\mathcal{L}$ and $\mathcal{K}$. The class $\mathcal{L}$ stands out for its canonicity: it results from Curry-Howard interpretations of classical linear logic propositions as session types. The class $\mathcal{K}$, obtained by encoding session types into Kobayashi's linear types with usages, includes processes not typable in other type systems. We show that $\mathcal{L}$ is strictly included in $\mathcal{K}$, and identify the precise conditions under which they coincide. We also provide two type-preserving translations of processes in $\mathcal{K}$ into processes in $\mathcal{L}$.

© 2021 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

In this paper, we formally relate different type systems for concurrent processes specified in the π-calculus [45]. A fundamental model of computation, the π-calculus stands out for its expressiveness, which enables us to represent and reason about message-passing programs in functional, object-oriented, and distributed paradigms [54]. Another distinctive aspect of the π-calculus is its support for rich type systems that discipline process behaviour [53]. Following Milner's seminal work on *sorting* [44], various type systems for the π-calculus have revealed a rich landscape of models for concurrency with disciplined communication; examples include *graph types* [62], *linear types* [40], *generic types* [30], and *session types* [28,56,29]. In the last decade, logical foundations for message-passing concurrency, in the style of Curry-Howard correspondence ("propositions as types") between linear logic and session types, have been put forward [9,61]. By disciplining the use of channels, types for message-passing processes strongly influence their expressiveness. Contrasting different type systems through the classes of well-typed processes that they induce is a central theme in this work.

Our interest is in *session-based concurrency*, the model of concurrency captured by session types. Session types promote a type-based approach to communication correctness: dialogues between participants are structured into *sessions*, basic communication units; descriptions of interaction sequences are then abstracted as session types which are checked against process specifications. In session-based concurrency, types enforce correct communications through different safety and

* Corresponding authors.
E-mail addresses: Ornela.Dardha@glasgow.ac.uk (O. Dardha), j.a.perez@rug.nl (J.A. Pérez).

https://doi.org/10.1016/j.jlamp.2021.100717
2352-2208/© 2021 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

liveness properties. Two basic (and intertwined) correctness properties are *communication safety* and *session fidelity*: while the former ensures absence of errors (e.g., communication mismatches), the latter ensures that well-typed processes respect the protocols prescribed by session types.

A very desirable liveness property for safe processes is that they should never "get stuck". This is the well-known *progress* property, which asserts that a well-typed term either is a final value or can further reduce [51]. In calculi for concurrency, this property admits several formalizations; two of them are *deadlock freedom* and *lock freedom*. Intuitively, deadlock freedom ensures that communications will eventually succeed unless the whole process diverges [38]; lock freedom is a stronger property: it guarantees that communications will eventually succeed, regardless of whether processes diverge, modulo some fairness assumption on the scheduler [36]. Notice that in the absence of divergent behaviours, deadlock freedom and lock freedom coincide.

Another formalization, which we call here *session progress*, has been proposed for session-based concurrency [7,8]: "a process has session progress if combined with another process providing the corresponding co-actions (a so-called *catalyzer*), then the composition reduces". We will say that a process is *composable* if it can be composed with an appropriate catalyzer for some of its actions, or *uncomposable* otherwise. Carbone et al. [8] proved that session progress and (dead)lock freedom coincide for uncomposable processes; for composable processes, session progress states *potential* (dead)lock freedom. We will return to this informal distinction between composable and uncomposable processes below (Definition 2.2 will make it formal).

There is a vast literature on type systems for which typability enforces (dead)lock freedom or session progress—see, e.g., [36,38,15,7,9,11,46,60,47,24,33,16,5]. Unfortunately, these sophisticated systems rely on different principles and/or consider different variants of the (session) π-calculus. Also, different papers sometimes use different terminology and notions. As a result, very little is known about the relationship between these type systems. These observations led us to our research questions: *How do the type systems for liveness properties relate to each other? What classes of deadlock-free processes do they induce?*

This paper presents the *first formal comparison* between different type systems for the π-calculus that enforce liveness properties related to (dead)lock freedom. More concretely, we tackle the above open questions by comparing $\mathcal{L}$ and $\mathcal{K}$, two salient classes of deadlock-free (session) typed processes:

- The class $\mathcal{L}$ contains session processes that are well-typed under the Curry-Howard correspondence between (classical) linear logic propositions and session types [9,11,61]. This suffices, because the type system derived from such a correspondence simultaneously ensures communication safety, session fidelity, and deadlock freedom.
- The class $\mathcal{K}$ contains session processes that enjoy communication safety and session fidelity (as ensured by the type system of Vasconcelos [57]) as well as satisfy deadlock freedom. This class of processes is defined indirectly, by combining Kobayashi's type system based on usages [36,38,39] with encodability results by Dardha et al. [17].

Let us clarify the nature of processes in $\mathcal{L}$ and $\mathcal{K}$. As Definition 4.4 formally states, processes in $\mathcal{L}$ and $\mathcal{K}$ are typed under some typing context, possibly non empty. As such, these classes contain both composable processes (if the typing context is not empty) and uncomposable processes (otherwise). Thus, strictly speaking, processes in $\mathcal{L}$ and $\mathcal{K}$ have session progress (as described above), which is strictly weaker than deadlock freedom, because a process satisfying session progress does not necessarily satisfy deadlock freedom. This is trivially true for composable processes, which need a catalyzer [8]. However, since we shall focus on uncomposable processes, for which session progress and deadlock freedom coincide, we shall refer to $\mathcal{L}$ and $\mathcal{K}$ as classes of deadlock-free processes.

There are good reasons for investigating $\mathcal{L}$ and $\mathcal{K}$. On one hand, due to its deep logical foundation, the class $\mathcal{L}$ appears to us as the canonical class of deadlock-free session processes, upon which all other classes should be compared. Indeed, this class arguably offers the most principled yardstick for comparisons. On the other hand, the class $\mathcal{K}$ integrates session type checking with the sophisticated usage discipline developed by Kobayashi for π-calculus processes. This indirect approach to deadlock freedom, developed in [8,14], is general: it can capture sessions with subtyping, polymorphism, and higher-order communication. Also, as discussed in [8], $\mathcal{K}$ strictly includes classes of session typed processes induced by other type systems for deadlock freedom [15,7,46].

Contributions. This paper contributes technical results that, on the one hand, *separate* the classes $\mathcal{L}$ and $\mathcal{K}$ by precisely characterizing the fundamental differences between them and, on the other hand, *unify* these classes by showing how their differences can be overcome to translate processes in $\mathcal{K}$ into processes into $\mathcal{L}$. More in details:

- To *separate* $\mathcal{L}$ from $\mathcal{K}$, we define $\mu\mathcal{K}$: a sub-class of $\mathcal{K}$ whose definition internalizes the key aspects of the Curry-Howard interpretations of session types. In particular, $\mu\mathcal{K}$ adopts the principle of "composition plus hiding", a distinguishing feature of the interpretations in [9,61], by which concurrent cooperation is restricted to the sharing of *exactly one* session channel.
  We show that $\mathcal{L}$ and $\mu\mathcal{K}$ coincide (Theorem 4.1). This gives us a separation result: there are deadlock-free session processes that *cannot* be typed by systems derived from the Curry-Howard interpretation of session types [9,11,61], but that are admitted as typable by the (indirect) approach of [8,14].

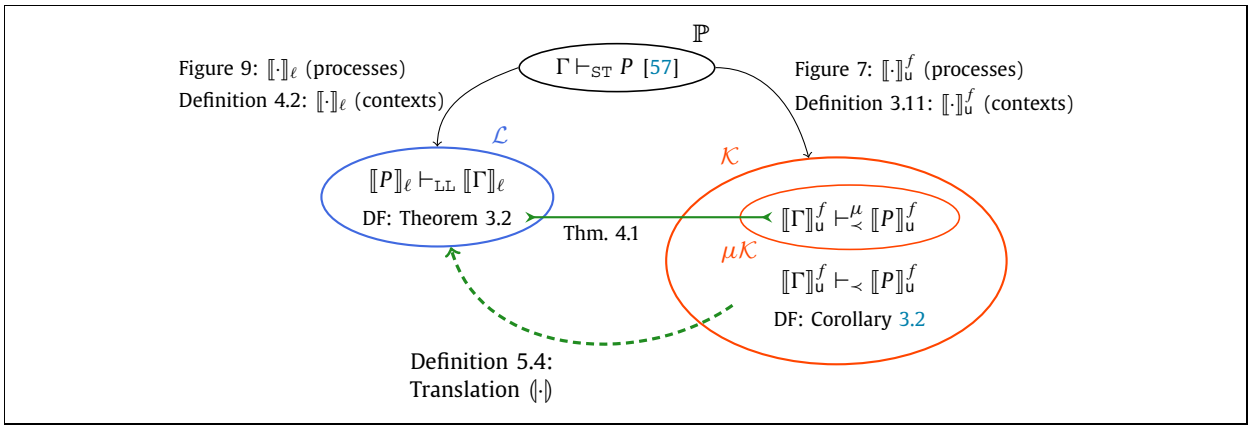


Fig. 1. Summary of type systems, languages with deadlock freedom (DF), and encodings between them (indicated by solid black arrows). Main results are denoted by green lines: our separation result, based on the coincidence of $\mathcal{L}$ and $\mu\mathcal{K}$ is indicated by the solid line with reversed arrowheads; our unifying result is indicated by the dashed arrow. (If the figure above does not feature colours, please referred to the web version of this article.)

- To *unify* $\mathcal{L}$ and $\mathcal{K}$, we define two *translations* of processes in $\mathcal{K}$ into processes in $\mathcal{L}$ (Definition 5.4). Intuitively, because the difference between $\mathcal{L}$ and $\mathcal{K}$ lies in the forms of parallel composition they admit (restricted in $\mathcal{L}$, liberal in $\mathcal{K}$), it is natural to transform a process in $\mathcal{K}$ into another, more parallel process in $\mathcal{L}$. In essence, the first translation, denoted $(\!|\cdot|\!)$ (Definition 5.4), exploits type information to replace sequential prefixes with representative parallel components; the second translation refines this idea by considering *value dependencies*, i.e., causal dependencies between independent sessions not captured by types. We detail the first translation, which satisfies type-preservation and operational correspondence properties (Theorems 5.1 and 5.2).

Our separation result is significant as it establishes the precise status of logically motivated systems with respect to previous disciplines, not based on Curry-Howard principles. It also provides a new characterization for linear logic-based processes, leveraging a very different type system for deadlock freedom [38]. Also, our unifying result is insightful because it shows that the differences between the two classes of processes manifest themselves rather subtly at the level of process syntax, and can be eliminated by appropriately exploiting information at the level of types.

To further illustrate these salient points, consider the process

$$P \triangleq (\boldsymbol{\nu} xy)(\boldsymbol{\nu} wz)(x(s).w\langle s\rangle \mid \overline{y}\langle \mathbf{n}\rangle.\overline{z}(u))$$

where, following the syntax of the type system of Vasconcelos [57], we use the restriction $(\boldsymbol{\nu} xy)$ to indicate that x and y are the two endpoints of the same channel (and similarly for $(\boldsymbol{\nu} wz)$). Also, $\mathbf{n}$ denotes a terminated channel. Process P belongs to the class $\mathcal{K}$. It consists of two processes in parallel, each implementing *two separate sessions* in sequence; as such, P does not respect the monolithic structure imposed by "composition plus hiding". Our separation result is that P falls outside $\mu\mathcal{K}$, the strict sub-class of $\mathcal{K}$ that adheres to "composition plus hiding". Now, process P has an operationally equivalent cousin, the process

$$Q \triangleq (\boldsymbol{\nu} xy)(\boldsymbol{\nu} wz)(x(s).w\langle s\rangle \mid \overline{y}\langle \mathbf{n}\rangle \mid \overline{z}(u))$$

which is structurally equivalent to one that is more parallel and respects "composition plus hiding":

$$Q \equiv (\boldsymbol{\nu} wz)((\boldsymbol{\nu} xy)(x(s).w\langle s\rangle \mid \overline{y}\langle \mathbf{n}\rangle) \mid \overline{z}(u))$$

Our unifying result is that all processes in $\mathcal{K}$ but outside of $\mu\mathcal{K}$ have these kind of more parallel, operationally equivalent cousins, which respect the "composition plus hiding" principle that characterizes class $\mathcal{L}$.

To our knowledge, our work is the first to formally compare fundamentally distinct type systems for deadlock freedom in message-passing processes. Previous comparisons by Carbone et al. [8] and Caires et al. [11, §6], are informal: they are based on representative "corner cases", namely examples of deadlock-free session processes typable in one system but not in some other.

Fig. 1 summarizes the different type systems and ingredients needed to define $\mathcal{K}$ and $\mathcal{L}$.

Paper organization. In § 2 we summarize the session π-calculus and the type system of [57]. In § 3 we present the two typed approaches to deadlock freedom for sessions. § 4 defines the classes $\mathcal{L}$ and $\mathcal{K}$, the strict sub-class $\mu\mathcal{K}$, and shows that $\mathcal{L} = \mu\mathcal{K}$ (Theorem 4.1). § 5 presents our first translation of $\mathcal{K}$ into $\mathcal{L}$ and establishes its correctness properties. The optimization with value dependencies is discussed in § 5.3. Enforcing deadlock freedom by typing is already challenging

$$
\begin{array}{rrlll}
P, Q ::= & \overline{x}\langle v\rangle.P & \text{(output)} & \mid \ \mathbf{0} & \text{(inaction)} \\
\mid & x(y).P & \text{(input)} & \mid \ P \mid Q & \text{(composition)} \\
\mid & x \lhd l_j.P & \text{(selection)} & \mid \ (\boldsymbol{\nu} xy)P & \text{(session restriction)} \\
\mid & x \rhd \{l_i : P_i\}_{i\in I} & \text{(branching)} & & \\
v ::= & x & \text{(channel)} & &
\end{array}
$$

$$
\begin{array}{ll}
\text{(R-Com)} & (\boldsymbol{\nu} xy)(\overline{x}\langle v\rangle.P \mid y(z).Q) \to (\boldsymbol{\nu} xy)(P \mid Q[v/z]) \\
\text{(R-Case)} & (\boldsymbol{\nu} xy)(x \lhd l_j.P \mid y \rhd \{l_i : P_i\}_{i\in I}) \to (\boldsymbol{\nu} xy)(P \mid P_j) \quad j \in I \\
\text{(R-Par)} & P \to Q \Longrightarrow P \mid R \to Q \mid R \\
\text{(R-Res)} & P \to Q \Longrightarrow (\boldsymbol{\nu} xy)P \to (\boldsymbol{\nu} xy)Q \\
\text{(R-Str)} & P \equiv P',\ P \to Q,\ Q' \equiv Q \Longrightarrow P' \to Q'
\end{array}
$$

Fig. 2. The session π-calculus: syntax (top) and semantics (bottom).

for processes without constructs such as recursion or replication. For this reason, here we concentrate on finite processes; in § 6 we discuss the case of processes with unbounded behaviour (with constructs such as replication and recursion) and connections with other logical interpretations of session types. § 7 compares with related works and § 8 collects some concluding remarks. Omitted technical details are included in the appendices.

This paper is a revised version of the workshop paper [19], extended with new material: we present full technical details and additional examples not presented in [19]. Also, we present updated comparisons with related works. The separation result based on the coincidence with the sub-class $\mu\mathcal{K}$, given in § 4, is new. Moreover, the first translation, presented in [19] and given in § 5, has been substantially simplified and its correctness properties have been refined. The content of § 6 is also original to this presentation.

2. Session π-calculus

We present the session π-calculus and its corresponding type system: the linear fragment of the type system by Vasconcelos [57], which ensures communication safety and session fidelity (but not progress nor deadlock freedom). Below we follow the definitions and results from [57], pointing out differences where appropriate.

2.1. Process model

Let $P, Q, \ldots$ range over processes, $x, y, \ldots$ over channel names (or *session endpoints*), and $v, v', \ldots$ over values; for simplicity, the sets of values and channels coincide. In examples, we use $\mathbf{n}$ to denote a terminated channel that cannot be further used.

We briefly comment on the syntax of processes, given in Fig. 2 (upper part). The main difference with respect to the syntax in [57] is that we do not consider boolean values nor conditional processes (if-then-else). Process $\overline{x}\langle v\rangle.P$ denotes the output of v along x, with continuation P. Dually, process $x(y).P$ denotes an input along x with continuation P, with y denoting a placeholder. Rather than the non-deterministic choice of the untyped π-calculus [45], the session π-calculus includes operators for (deterministic) internal and external labelled choices, denoted $x \lhd l_j.P$ and $x \rhd \{l_i : P_i\}_{i\in I}$, respectively. Process $x \lhd l_j.P$ uses x to select l_j from a labelled choice process $x \rhd \{l_i : P_i\}_{i\in I}$, so as to trigger P_j; labels are indexed by the finite set I and are pairwise distinct. We also have the inactive process (denoted $\mathbf{0}$), the parallel composition of P and Q (denoted $P \mid Q$), and the *double restriction* operator, noted $(\boldsymbol{\nu} xy)P$: the intention is that x and y denote *dual* session endpoints with scope P. We omit $\mathbf{0}$ whenever possible and write, e.g., $\overline{x}\langle\mathbf{n}\rangle$ instead of $\overline{x}\langle\mathbf{n}\rangle.\mathbf{0}$. We will write $\mathbb{P}$ to denote the session π-calculus processes generated by the grammar in Fig. 2.

Notions of free and bound names in processes are exactly as in [57]. That is, name y is bound in $x(y).P$ and names x and y are bound in $(\boldsymbol{\nu} xy)P$. A name that does not occur bound within a process is said to be free; the set of free names of P is denoted $\mathrm{fn}(P)$. We write $P[v/z]$ to denote the (capture-avoiding) substitution of free occurrences of z in P with v. Finally, we follow *Barendregt's variable convention*, whereby all names in binding occurrences in any mathematical context are pairwise distinct and also distinct from the free names.

The operational semantics is given in terms of a reduction relation, noted $P \to Q$, and defined by the rules in Fig. 2 (lower part). Reduction relies on a standard notion of *structural congruence*, noted $\equiv$, defined by the following axioms:

$$
\begin{array}{llll}
P \mid Q \equiv Q \mid P & (P \mid Q) \mid R \equiv P \mid (Q \mid R) & P \mid \mathbf{0} \equiv P & \\
(\boldsymbol{\nu} xy)P \mid Q \equiv (\boldsymbol{\nu} xy)(P \mid Q) & (\boldsymbol{\nu} xy)\mathbf{0} \equiv \mathbf{0} & (\boldsymbol{\nu} wx)(\boldsymbol{\nu} yz)P \equiv (\boldsymbol{\nu} yz)(\boldsymbol{\nu} wx)P &
\end{array}
$$

$$\text{(T-Nil)}\ \frac{\mathsf{un}(\Gamma)}{\Gamma \vdash_{\mathrm{ST}} \mathbf{0}} \qquad \text{(T-Par)}\ \frac{\Gamma_1 \vdash_{\mathrm{ST}} P \quad \Gamma_2 \vdash_{\mathrm{ST}} Q}{\Gamma_1, \Gamma_2 \vdash_{\mathrm{ST}} P \mid Q} \qquad \text{(T-Res)}\ \frac{\Gamma, x : T, y : \overline{T} \vdash_{\mathrm{ST}} P}{\Gamma \vdash_{\mathrm{ST}} (\nu xy)P}$$

$$\text{(T-In)}\ \frac{\Gamma, x : S, y : T \vdash_{\mathrm{ST}} P}{\Gamma, x : ?T.S \vdash_{\mathrm{ST}} x(y).P} \qquad \text{(T-Out)}\ \frac{\Gamma, x : S \vdash_{\mathrm{ST}} P}{\Gamma, x : !T.S, y : T \vdash_{\mathrm{ST}} \overline{x}\langle y\rangle.P}$$

$$\text{(T-Brch)}\ \frac{\Gamma, x : S_i \vdash_{\mathrm{ST}} P_i \quad \forall i \in I}{\Gamma, x : \&\{l_i : S_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleright \{l_i : P_i\}_{i\in I}} \qquad \text{(T-Sel)}\ \frac{\Gamma, x : S_j \vdash_{\mathrm{ST}} P \quad \exists j \in I}{\Gamma, x : \oplus\{l_i : S_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleleft l_j.P}$$

Fig. 3. Typing rules for the session π-calculus.

Key rules in Fig. 2 are (R-Com) and (R-Case), denoting the interaction of output/input prefixes and selection/branching constructs, respectively. Observe that interaction involves prefixes with different channels (endpoints), and always occurs in the context of an outermost double restriction. Rules (R-Par), (R-Res), and (R-Str) are standard [57]. We write $\rightarrow^*$ to denote the reflexive, transitive closure of $\rightarrow$.

2.2. Type system

The syntax of session types, ranged over $T, S, \ldots$, is given by the following grammar.

$$T, S ::= ?T.S \mid !T.S \mid \&\{l_i : S_i\}_{i\in I} \mid \oplus\{l_i : S_i\}_{i\in I} \mid \mathbf{end}$$

The type $?T.S$ is assigned to an endpoint that first receives a value of type T and then continues according to the protocol described by S. Dually, type $!T.S$ is assigned to an endpoint that first outputs a value of type T and then continues according to the protocol described by S. Type $\&\{l_i : S_i\}_{i\in I}$ is used for external choices, and generalises input types; dually, type $\oplus\{l_i : S_i\}_{i\in I}$ is used for internal choices, and generalises output types. Finally, **end** is the type of an endpoint with a terminated protocol. Notice that session types describe protocols as *sequences* of actions; they do not admit parallel operators.

With respect to the syntax of types in [57], we only consider channel endpoint types (no ground types such as bool). Also, types in the system in [57] can be qualified as either *linear* or *unrestricted*. Our session types are linear—the only unrestricted session type is **end**. Focusing on linear types suffices for our purposes, and leads to simplifications in typing rules and auxiliary notions, such as well-formedness (see below).

A central notion in session-based concurrency is *duality*, which relates session types offering opposite (i.e., complementary) behaviours; it stands at the basis of communication safety and session fidelity. Given a session type T, its dual type $\overline{T}$ is defined as follows:

$$\begin{array}{rcl} \overline{\mathbf{end}} & \triangleq & \mathbf{end} \\ \overline{!T.S} & \triangleq & ?T.\overline{S} \\ \overline{?T.S} & \triangleq & !T.\overline{S} \\ \overline{\oplus\{l_i : S_i\}_{i\in I}} & \triangleq & \&\{l_i : \overline{S}_i\}_{i\in I} \\ \overline{\&\{l_i : S_i\}_{i\in I}} & \triangleq & \oplus\{l_i : \overline{S}_i\}_{i\in I} \end{array}$$

Typing contexts, ranged over by Γ, Γ', are produced by the following grammar:

$$\Gamma, \Gamma' ::= \emptyset \mid \Gamma, x : T$$

where '$\emptyset$' denotes the empty context. We standardly require the variables appearing in a context to be pairwise distinct. It is often convenient to treat typing contexts as sets of typing assignments $x : T$. This way, e.g., we write $x : T \in \Gamma$ if $\Gamma = \Gamma', x : T$, for some Γ'. We write $\mathsf{un}(\Gamma)$ if and only if $x : T \in \Gamma$ implies $T = \mathbf{end}$. We sometimes write Γ^{un} to indicate that $\mathsf{un}(\Gamma)$. Given a context Γ and a process P, a session typing judgement is of the form $\Gamma \vdash_{\mathrm{ST}} P$. If Γ is empty, we write $\vdash_{\mathrm{ST}} P$.

Typing rules are given in Fig. 3. Rule (T-Nil) states that $\mathbf{0}$ is only well-typed under a fully terminated context. Rule (T-Par) types the parallel composition of two processes by composing their corresponding typing contexts.[1] Rule (T-Res) types a restricted process by requiring that the two endpoints have dual types. Rules (T-In) and (T-Out) type the receiving and

[1] In the presence of unrestricted types, as given in [57], Rule (T-Par) requires a *splitting operator*, noted $\circ$ in [57]. However, since we consider only linear session types, the $\circ$ operator boils down to ','.

sending of a value over a channel x, respectively. Finally, Rules (T-Brch) and (T-Sel) are generalizations of input and output over a labelled set of processes.

The main guarantees of the type system in [57] are *communication safety* and *session fidelity*, i.e., typed processes respect their ascribed protocols, as represented by session types. We have the following results:

Theorem 2.1 *(Strengthening – Lemma 7.3 in [57]). Let* $\Gamma \vdash_{\mathtt{ST}} P$ *and* $x \notin \mathsf{fn}(P)$. *If* $\Gamma = \Gamma', x : T$ *then* $\Gamma' \vdash_{\mathtt{ST}} P$.

Theorem 2.2 *(Preservation for* $\equiv$ *– Lemma 7.4 in [57]). If* $\Gamma \vdash_{\mathtt{ST}} P$ *and* $P \equiv Q$, *then* $\Gamma \vdash_{\mathtt{ST}} Q$.

Theorem 2.3 *(Preservation – Theorem 7.2 in [57]). If* $\Gamma \vdash_{\mathtt{ST}} P$ *and* $P \to Q$, *then* $\Gamma \vdash_{\mathtt{ST}} Q$.

Following [57], we say that processes of the form $\overline{x}\langle v\rangle.P$, $x(z).Q$, $x \triangleleft l_j.P$, and $x \triangleright \{l_i : P_i\}_{i\in I}$ are *prefixed at* x. We call *redexes* processes of the form $\overline{x}\langle v\rangle.P \mid y(z).Q$ and $x \triangleleft l_j.P \mid y \triangleright \{l_i : P_i\}_{i\in I}$. The following notion of well-formed processes, a specialization of the definition in [57], is key to single out meaningful typed processes:

Definition 2.1 *(Well-formedness).* A process is *well-formed* if for each of its structural congruent processes of the form $(\boldsymbol{\nu} x_1 y_1)\ldots(\boldsymbol{\nu} x_n y_n)(P \mid Q \mid R)$ the following condition holds.

- If P is prefixed at x_1 and Q is prefixed at y_1 then $P \mid Q$ is a redex.

We have the following result:

Theorem 2.4 *(Safety – Theorem 7.3 in [57]). If* $\vdash_{\mathtt{ST}} P$ *then* P *is well-formed.*

Therefore, if $\vdash_{\mathtt{ST}} P$ and P reduces to Q in zero or more steps, then Q is well-formed; this is Theorem 7.1 in [57].
We close by introducing some useful terminology:

Definition 2.2 *((Un)composable processes).* Let $\Gamma \vdash_{\mathtt{ST}} P$. If Γ is empty, we say that P is *uncomposable*; otherwise, if Γ is non-empty, we say P is *composable*.

We use the adjectives composable and uncomposable to distinguish typable processes depending on their associated typing context. Note that the adjectives *open* and *closed* have their usual meaning, associated to the free names of a process, possibly untyped. As an example, process $\mathbf{0}$ is a closed process that can be typed under a non-empty typing context, and so it is composable.

2.3. Deadlock freedom

As already motivated, a desirable liveness property for session π-calculus processes is that they should never "get stuck". Unfortunately, the session type system given in [57] (and summarized above) does not exclude deadlocked processes. Intuitively, this is because typed processes may contain cyclic causal dependencies enforced by communication prefixes in processes but not described by their session types. Indeed, a particularly insidious class of deadlocks is due to cyclic interleaving of channels in processes, as illustrated by following example.

Example 2.1 *(A deadlocked process).* Process $P \triangleq (\boldsymbol{\nu} xy)(\boldsymbol{\nu} wz)(\overline{x}\langle \mathbf{n}\rangle.\overline{w}\langle \mathbf{n}\rangle \mid z(t).y(s))$ represents the implementation of two independent sessions, xy and wz, which get intertwined (blocked) due to the nesting induced by input and output prefixes. Process P is well-typed in [57] under $\mathbf{n} : \mathbf{end} \vdash_{\mathtt{ST}} P$, even if P is unable to reduce.

Below we define deadlock freedom in the session π-calculus by following [33, Def. 5.2]:

Definition 2.3 *(Deadlock freedom).* A process P is *deadlock-free* if the following condition holds: whenever $P \to^* P'$ and one of the following holds

- $P' \equiv (\boldsymbol{\nu} \widetilde{xy})(\overline{x}\langle v\rangle.Q_1 \mid Q_2)$
- $P' \equiv (\boldsymbol{\nu} \widetilde{xy})(x(y).Q_1 \mid Q_2)$
- $P' \equiv (\boldsymbol{\nu} \widetilde{xy})(x \triangleleft l_j.Q_1 \mid Q_2)$
- $P' \equiv (\boldsymbol{\nu} \widetilde{xy})(x \triangleright \{l_i : P_i\}_{i\in I} \mid Q)$

then there exists R such that $P' \to R$.

$$
\begin{array}{rl}
\text{(R-ChCom)} & \overline{x}\langle v\rangle.P \mid x(z).Q \to P \mid Q\,[^{v}/_{z}] \\
\text{(R-Fwd)} & (\boldsymbol{\nu} x)([x\leftrightarrow y] \mid P) \to P[^{y}/_{x}] \\
\text{(R-ChCase)} & x \triangleleft l_j.P \mid x \triangleright \{l_i : P_i\}_{i\in I} \to P \mid P_j \quad j \in I \\
\text{(R-ChRes)} & P \to Q \Longrightarrow (\boldsymbol{\nu} x)P \to (\boldsymbol{\nu} x)Q
\end{array}
$$

Fig. 4. Reduction rules for processes in $\mathcal{L}$.

Remark 1 *(Defining deadlock freedom).* Definition 2.3 is closely related to the definition of deadlock and deadlock freedom in [36] (Definition 2.4), which states that a process P is in deadlock if it reaches one of the first two items stated in Definition 2.3 and cannot reduce from there. Then, a process P defined as deadlock-free if it never reduces to a deadlocked process. We shall be following the type system in [38], where the notion of deadlock freedom is defined informally: a process is deadlock-free if when "given a request, it will eventually return a result unless the process diverges".

Example 2.2 *(A deadlock-free process).* It is easy to see that process P from Example 2.1 is not deadlock-free as per Definition 2.3. A deadlock-free variant of process P would be $P' \triangleq (\boldsymbol{\nu} xy)(\boldsymbol{\nu} wz)(\overline{x}\langle \mathbf{n}\rangle.\overline{w}\langle \mathbf{n}\rangle \mid y(s).z(t))$, which also is typable: $\mathbf{n} : \mathbf{end} \vdash_{\mathrm{ST}} P'$. Observe how the difference between P and P' is in the parallel component on the right-hand side: the two input prefixes have been swapped.

3. Two approaches to deadlock freedom

We introduce two approaches to typing deadlock-free processes. The first comes from interpretations of linear logic propositions as session types [9,11,61] (§ 3.1). The second approach exploits encodings of session processes and types [17] into the linear types with *usages* for the π-calculus (§ 3.2). Based on these approaches, in § 4 we will formally define the classes $\mathcal{L}$ and $\mathcal{K}$ mentioned in the introduction.

3.1. Linear logic foundations of session types

The linear logic interpretation of session types was introduced by Caires and Pfenning [9,11], and developed by Wadler [61] and others. Here we consider an interpretation based on classical linear logic (CLL) with mix principles, following [6,10].

The syntax and semantics of processes are as in § 2 with the following differences. First, we have a single restriction construct $(\boldsymbol{\nu} x)P$ instead of the double restriction $(\boldsymbol{\nu} xy)P$. Second, we have a *forwarding process*, denoted $[x\leftrightarrow y]$, which intuitively "fuses" or "links" channels/names x and y. More formally, we have:

$$P, Q ::= \overline{x}\langle v\rangle.P \mid x(y).P \mid x \triangleleft l_j.P \mid x \triangleright \{l_i : P_i\}_{i\in I} \mid (\boldsymbol{\nu} x)P \mid [x\leftrightarrow y] \mid P \mid Q \mid \mathbf{0}$$

In what follows, the bound output $(\boldsymbol{\nu} y)\overline{x}\langle y\rangle.P$ will be abbreviated as $\overline{x}(y)P$. Also, we write $(\boldsymbol{\nu}\widetilde{x})P$ to abbreviate $(\boldsymbol{\nu} x_1)\ldots(\boldsymbol{\nu} x_n)P$.

Differences in the reduction rules are summarized in Fig. 4. In particular, observe how interaction of input/output prefixes and of selection/branching constructs is no longer covered by an outermost restriction.

As for the type system, we consider the so-called *linear logic types* which correspond to linear logic propositions (without exponentials). They are given by the following grammar:

$$A, B ::= \bot \mid \mathbf{1} \mid A \otimes B \mid A \parr B \mid \&\{l_i : A_i\}_{i\in I} \mid \oplus\{l_i : A_i\}_{i\in I}$$

Intuitively, $\bot$ and $\mathbf{1}$ are used to type a terminated endpoint. Type $A \otimes B$ is associated to an endpoint that first outputs an object of type A and then behaves according to B. Dually, type $A \parr B$ is the type of an endpoint that first inputs an object of type A and then continues as B. The interpretation of $\&\{l_i : A_i\}_{i\in I}$ and $\oplus\{l_i : A_i\}_{i\in I}$ as types for branching and selection behaviours is precisely as in session types (cf. § 2.2).

A full duality on linear logic types corresponds to the negation operator of CLL $(\cdot)^{\perp}$. The *dual* of type A, denoted $\overline{A}$, is inductively defined as follows:

$$
\begin{array}{rcl}
\overline{\mathbf{1}} & \triangleq & \bot \\
\overline{\bot} & \triangleq & \mathbf{1} \\
\overline{A \otimes B} & \triangleq & \overline{A} \parr \overline{B} \\
\overline{A \parr B} & \triangleq & \overline{A} \otimes \overline{B} \\
\overline{\&\{l_i : A_i\}_{i\in I}} & \triangleq & \oplus\{l_i : \overline{A_i}\}_{i\in I} \\
\overline{\oplus\{l_i : A_i\}_{i\in I}} & \triangleq & \&\{l_i : \overline{A_i}\}_{i\in I}
\end{array}
$$

$$\text{(T-}\mathbf{1}\text{)}\ \frac{}{\mathbf{0} \vdash_{\mathrm{LL}} x{:}\bullet} \qquad \text{(T-}\bot\text{)}\ \frac{P \vdash_{\mathrm{LL}} \Delta}{P \vdash_{\mathrm{LL}} x{:}\bullet, \Delta} \qquad \text{(T-id)}\ \frac{}{[x \leftrightarrow y] \vdash_{\mathrm{LL}} x{:}A,\ y{:}\overline{A}}$$

$$\text{(T-}\parr\text{)}\ \frac{P \vdash_{\mathrm{LL}} \Delta, y{:}A, x{:}B}{x(y).P \vdash_{\mathrm{LL}} \Delta, x{:}A \parr B} \qquad \text{(T-}\otimes\text{)}\ \frac{P \vdash_{\mathrm{LL}} \Delta, y{:}A \qquad Q \vdash_{\mathrm{LL}} \Delta', x{:}B}{\overline{x}(y).(P \mid Q) \vdash_{\mathrm{LL}} \Delta, \Delta', x{:}A \otimes B}$$

$$\text{(T-}\oplus\text{)}\ \frac{P \vdash_{\mathrm{LL}} \Delta, x{:}A_j \qquad j \in I}{x \triangleleft l_j.P \vdash_{\mathrm{LL}} \Delta, x{:} \oplus \{l_i : A_i\}_{i \in I}} \qquad \text{(T-\&)}\ \frac{P_i \vdash_{\mathrm{LL}} \Delta, x{:}A_i \qquad \forall i \in I}{x \triangleright \{l_i : P_i\}_{i \in I} \vdash_{\mathrm{LL}} \Delta, x{:}\&\{l_i : A_i\}_{i \in I}}$$

$$\text{(T-cut)}\ \frac{P \vdash_{\mathrm{LL}} \Delta, x{:}\overline{A} \qquad Q \vdash_{\mathrm{LL}} \Delta', x{:}A}{(\boldsymbol{\nu} x)(P \mid Q) \vdash_{\mathrm{LL}} \Delta, \Delta'} \qquad \text{(T-mix)}\ \frac{P \vdash_{\mathrm{LL}} \Delta \qquad Q \vdash_{\mathrm{LL}} \Delta'}{P \mid Q \vdash_{\mathrm{LL}} \Delta, \Delta'}$$

Fig. 5. Typing rules for the π-calculus with linear logic types.

Recall that $A \multimap B \triangleq \overline{A} \parr B$. As explained in [6], considering mix principles means admitting $\bot \multimap \mathbf{1}$ and $\mathbf{1} \multimap \bot$, and therefore $\bot = \mathbf{1}$. We write $\bullet$ to denote either $\bot$ or $\mathbf{1}$, and therefore $\overline{\bullet} = \bullet$. That is, we consider the *conflation* of dual types $\bot$ and $\mathbf{1}$ as explored by Atkey et al. [2].

Typing contexts, ranged over $\Delta, \Delta', \ldots$, are produced by the following grammar:

$$\Delta, \Delta' ::= \cdot \ \mid\ \Delta, x : A$$

where '$\cdot$' denotes the empty typing context.

Typing judgements are of the form $P \vdash_{\mathrm{LL}} \Delta$. Fig. 5 gives the corresponding typing rules. One salient point is Rule (T-cut), which types two processes that have exactly one channel of dual type in common (x in the rule) by composing them in parallel and immediately restricting this common channel. This implements the *"composition plus hiding"* principle, which monolithically integrates parallel composition and restriction in a single rule. Indeed, unlike the system in [57], there is no dedicated rule for restriction, which also appears in the bound output induced by Rule (T-$\otimes$). Also, Rule (T-mix) conveniently enables to type the *independent* parallel composition of processes, i.e., the composition of two processes that do not share any sessions. (This is referred to as *communication-free concurrency* in [2].) Following Definition 2.2, we say that process $P \vdash_{\mathrm{LL}} \Delta$ is composable if $\Delta \neq \cdot$ and uncomposable otherwise.

We now collect main results for this type system; see [11,6,10] for details. We first state type preservation:

Theorem 3.1 *(Type preservation). If $P \vdash_{\mathrm{LL}} \Delta$ and $P \longrightarrow Q$ then $Q \vdash_{\mathrm{LL}} \Delta$.*

We now state deadlock freedom. For any P, define $live(P)$ if and only if $P \equiv (\boldsymbol{\nu} \widetilde{n})(\pi.Q \mid R)$, where π is an input, output, selection, or branching prefix.

Theorem 3.2 *(Deadlock freedom). If $P \vdash_{\mathrm{LL}} \cdot$ and $live(P)$ then $P \longrightarrow Q$, for some Q.*

3.2. Deadlock freedom by encodability

The second approach to deadlock-free session processes is *indirect*, in that establishing deadlock freedom for session processes appeals to encodings into a dyadic π-calculus whose type system enforces deadlock freedom by exploiting *usages*, *obligations*, and *capabilities* [36,38].

We follow closely the definitions and results in [38]. Next, we introduce the syntax of the (dyadic) π-calculus (§ 3.2.1), types with usages (§ 3.2.2), typing rules (§ 3.2.2), and finally the technical results leading to deadlock freedom (§ 3.2.3). The encodings of session processes into dyadic processes and of session types into types with usages are given in § 3.2.4.

3.2.1. The dyadic π-calculus

Syntax. The syntax of the dyadic π-calculus is as follows:

$$\begin{array}{rcll} v ::= & x & \text{(channel)} & \mid\ l_j\_v \quad \text{(variant value)} \\ P, Q ::= & \overline{x}\langle \widetilde{v} \rangle.P & \text{(output)} & \mid\ \mathbf{0} \quad \text{(inaction)} \\ \mid & x(\widetilde{z}).P & \text{(input)} & \mid\ P \mid Q \quad \text{(composition)} \\ \mid & \mathbf{case}\ v\ \mathbf{of}\ \{l_i\_x_i \triangleright P_i\}_{i \in I} & \text{(case)} & \mid\ (\boldsymbol{\nu} x)P \quad \text{(session restriction)} \end{array}$$

We discuss differences with respect to § 2 and [38]:

- While the syntax of processes given in § 2 (and in [38]) supports monadic communication, we consider *dyadic* communication: an output prefix involves a tuple of values v_1, v_2 and an input prefix involves a tuple of variables z_1, z_2. For the sake of notational uniformity, we write $\widetilde{v}$ and $\widetilde{z}$ to stand for v_1, v_2 and z_1, z_2, respectively. Dyadic communication is convenient, as the encoding of session processes into "standard" π-calculus processes [17] requires transmitting tuples whose length is at most two.
- While in [38] input and output prefixes are annotated with a capability annotation $t \in \mathbb{N} \cup \infty$, we omit such annotations to enhance clarity.
- Rather than the branching and selection constructs in § 2 (and the if-then-else process considered in [38]), in the dyadic π-calculus presented above we have the *case construct* $\mathbf{case}\, v\, \mathbf{of}\, \{l_i\_x_i \triangleright P_i\}_{i\in I}$, which uses the *variant value* l_j_v [54].
- We do not consider the let construct and replication processes in [38].
- In line with § 3.1 and [38] (but differently from § 2) we use $(\boldsymbol{\nu} x)P$ as restriction operator.

As before, we write $(\boldsymbol{\nu}\widetilde{x})P$ to denote the process $(\boldsymbol{\nu} x_1)\cdots(\boldsymbol{\nu} x_n)P$. Notions of free and bound names are as usual; we write $\mathsf{fn}(P)$ to denote the set of free names of P.

Reduction semantics. Following [38], the reduction semantics of dyadic π-calculus processes relies on the structural relation $\preceq$:

Definition 3.1 *(Def A2 in [38]).* The *structural relation* $\preceq$ is the least reflexive and transitive relation closed under the following rules (where $P \equiv Q$ denotes $(P \preceq Q) \wedge (Q \preceq P)$):

$$P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R)$$
$$P \mid \mathbf{0} \equiv P \qquad (\boldsymbol{\nu} x)P \mid Q \equiv (\boldsymbol{\nu} x)(P \mid Q) \quad \text{if } x \text{ is not free in } Q$$
$$(\boldsymbol{\nu} x)\mathbf{0} \equiv \mathbf{0} \qquad (\boldsymbol{\nu} x)(\boldsymbol{\nu} y)P \equiv (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)P$$
$$P \preceq Q \Longrightarrow P \mid R \preceq Q \mid R \qquad P \preceq Q \Longrightarrow (\boldsymbol{\nu} x)P \preceq (\boldsymbol{\nu} x)Q$$

Thus, $P \preceq Q$ intuitively means that P can be restructured into Q by using the above rules. We shall refer to $\equiv$ as structural equivalence. The reduction rules are then as follows:

$$(\text{R}\pi\text{-Com}) \quad \overline{x}\langle\widetilde{v}\rangle.P \mid x(\widetilde{z}).Q \to P \mid Q[\widetilde{v}/\widetilde{z}]$$
$$(\text{R}\pi\text{-Case}) \quad \mathbf{case}\, l_j\_v\, \mathbf{of}\, \{l_i\_x_i \triangleright P_i\}_{i\in I} \to P_j[v/x_j] \quad j \in I$$
$$(\text{R}\pi\text{-Par}) \quad P \to Q \Longrightarrow P \mid R \to Q \mid R$$
$$(\text{R}\pi\text{-Res}) \quad P \to Q \Longrightarrow (\boldsymbol{\nu} x)P \to (\boldsymbol{\nu} x)Q$$
$$(\text{R}\pi\text{-Str}) \quad P \preceq P',\ P \to Q,\ Q' \preceq Q \Longrightarrow P' \to Q'$$

Rules are self-explanatory. We only discuss Rule (Rπ-Case), which is the main difference with respect to the reduction semantics in [38]. Note that by using the variant value l_j_v, this rule simultaneously selects P_j *and* induces the substitution $[v/x_j]$. This is different from Rule (R-Case) for selection and branching (cf. Fig. 2), which selects a branch but does not involve a communication. As before, we write $\to^*$ to denote the reflexive, transitive closure of $\to$.

Since the definition of deadlock freedom in [38] is only informal (cf. Remark 1), we shall adopt the following definition, which mirrors Definition 2.3:

Definition 3.2 *(Deadlock freedom).* A process P is *deadlock-free* if the following condition holds: whenever $P \to^* P'$ and one of the following holds

- $P' \equiv (\boldsymbol{\nu}\widetilde{x})(\overline{x}\langle\widetilde{v}\rangle.Q_1 \mid Q_2)$
- $P' \equiv (\boldsymbol{\nu}\widetilde{x})(x(\widetilde{y}).Q_1 \mid Q_2)$
- $P' \equiv (\boldsymbol{\nu}\widetilde{x})(\mathbf{case}\, l_j\_v\, \mathbf{of}\, \{l_i\_x_i \triangleright P_i\}_{i\in I} \mid Q)$

then there exists R such that $P' \to R$.

3.2.2. Types with usages

The type system for deadlock freedom in [38] exploits types with usages. Usages rely on *obligations* and *capabilities*, which are endowed with *levels* to describe inter-channel dependencies:

- An obligation of level n must be fulfilled by using only capabilities of level *less than* n. Said differently, an action of obligation n may be prefixed by actions of capabilities less than n.

- For an action with capability of level n, there must exist a co-action with obligation of level *less than* n or *equal to* n.

We shall rely on usages as defined next, a strict subset of those defined in [38].

Definition 3.3 *(Usages).* The syntax of usages $U, U', \ldots$ is defined by the following grammar:

$$\begin{array}{lll} U ::= & 0 & \text{(not usable)} \\ \mid & ?^{\mathsf{o}}_{\kappa}.U & \text{(used in input)} \\ \mid & !^{\mathsf{o}}_{\kappa}.U & \text{(used in output)} \\ \mid & (U_1 \mid U_2) & \text{(used in parallel)} \\ \mid & \uparrow^t U & \text{(lift obligation levels of } U \text{ up to } t) \end{array}$$

where the *obligation* o and the *capability* κ range over the set $\mathbb{N} \cup \infty$. We shall refer to usages generated with the first three productions above (not usable, input, output) as *sequential usages*.

Usage 0 describes a channel that cannot be used at all. A usage $?^{\mathsf{o}}_{\kappa}.U$ (resp. $!^{\mathsf{o}}_{\kappa}.U$) is associated to a channel that can be used once for input (resp. output) and then according to usage U. The usage $U_1 \mid U_2$ can be associated to a channel that is used according to U_1 and U_2, possibly in parallel. The usage $\uparrow^t U$ acts as an operator that lifts the obligation levels in U up to t. We let α range over '?' and '!'. We will often omit 0, and so we will write, e.g., $\alpha^{\mathsf{o}}_{\kappa}$ instead of $\alpha^{\mathsf{o}}_{\kappa}.0$.

Notation 3.1 *(Co-actions).* We write $\overline{\alpha}$ to denote the *co-action* of α, i.e., $\overline{!} = ?$ and $\overline{?} = !$.

We rely on a number of auxiliary definitions for usages; they all follow [38]:

Definition 3.4 *(Capabilities and obligations).* Let U be a usage. The input and output *capability levels* (resp. *obligation levels*) of U, written $\mathsf{cap}_?(U)$ and $\mathsf{cap}_!(U)$ (resp. $\mathsf{ob}_?(U)$ and $\mathsf{ob}_!(U)$), are defined as:

$$\begin{array}{rcl@{\qquad}rcl} \mathsf{cap}_\alpha(0) & \triangleq & \infty & \mathsf{ob}_\alpha(0) & \triangleq & \infty \\ \mathsf{cap}_\alpha(\overline{\alpha}^{\mathsf{o}}_{\kappa}.U) & \triangleq & \infty & \mathsf{ob}_\alpha(\overline{\alpha}^{\mathsf{o}}_{\kappa}.U) & \triangleq & \infty \\ \mathsf{cap}_\alpha(\alpha^{\mathsf{o}}_{\kappa}.U) & \triangleq & \kappa & \mathsf{ob}_\alpha(\alpha^{\mathsf{o}}_{\kappa}.U) & \triangleq & \mathsf{o} \\ \mathsf{cap}_\alpha(U_1 \mid U_2) & \triangleq & \min(\mathsf{cap}_\alpha(U_1), \mathsf{cap}_\alpha(U_2)) & \mathsf{ob}_\alpha(U_1 \mid U_2) & \triangleq & \min(\mathsf{ob}_\alpha(U_1), \mathsf{ob}_\alpha(U_2)) \\ \mathsf{cap}_\alpha(\uparrow^t U) & \triangleq & \mathsf{cap}_\alpha(U) & \mathsf{ob}_\alpha(\uparrow^t U) & \triangleq & \max(t, \mathsf{ob}_\alpha(U)) \end{array}$$

We write $\mathsf{ob}(U)$ for $\max(\mathsf{ob}_?(U), \mathsf{ob}_!(U))$.

The reduction relation on usages, noted $U \to U'$, intuitively says that if a channel with usage U is used for communication then it should be used according to U' afterwards. It relies on an auxiliary structural relation on usages.

Definition 3.5 *(Structural relation on usages [38]).* Let $\preceq$ be the least reflexive and transitive relation on usages defined by the following rules:

$$U_1 \mid U_2 \preceq U_2 \mid U_1 \quad \uparrow^t (U_1 \mid U_2) \preceq (\uparrow^t U_1) \mid (\uparrow^t U_2) \quad (U_1 \mid U_2) \mid U_3 \preceq U_1 \mid (U_2 \mid U_3)$$
$$U_1 \preceq U_1' \wedge U_2 \preceq U_2' \Longrightarrow U_1 \mid U_1' \preceq U_2 \mid U_2' \quad \uparrow^t \alpha^{\mathsf{o}}_{\kappa}.U \preceq \alpha^{\max(\mathsf{o},t)}_{\kappa}.U \quad U \preceq U' \Longrightarrow \uparrow^t U \preceq \uparrow^t U'$$

Definition 3.6. The reduction relation $\to$ on usages is the smallest relation closed under the following rules:

$$\begin{array}{rl} \text{(U-Com)} & ?^{\mathsf{o}}_{\kappa}.U_1 \mid !^{\mathsf{o}'}_{\kappa'}.U_2 \to U_1 \mid U_2 \\ \text{(U-Par)} & U \to U' \Longrightarrow U \mid U'' \to U' \mid U'' \\ \text{(U-SubStruct)} & U \preceq U_1,\ U_1 \to U_2,\ U_2 \preceq U' \Longrightarrow U \to U' \end{array}$$

The reflexive, transitive closure of $\to$ (written $\to^*$) is defined as expected.

The following key definition ensures that if some action has a capability of level n then the obligation level of its co-actions should be at most n.

Definition 3.7 *(Reliability).* Let α and $\overline{\alpha}$ be co-actions (cf. Notation 3.1). We write $\mathsf{con}_\alpha(U)$ when $\mathsf{ob}_{\overline{\alpha}}(U) \preceq \mathsf{cap}_\alpha(U)$. We write $\mathsf{con}(U)$ when $\mathsf{con}_?(U)$ and $\mathsf{con}_!(U)$ hold. Usage U is *reliable*, noted $\mathsf{rel}(U)$, if $\mathsf{con}(U')$ holds for all U' such that $U \to^* U'$.

Example 3.1. We illustrate reliability and obligation/capability levels. Consider the usage $U = ?^{o_1}_{\kappa_1}.0 \mid !^{o_2}_{\kappa_2}.0$. We establish the conditions required for $\mathsf{rel}(U)$ to hold. This in turn requires determining a few ingredients:

- $\mathsf{ob}_!(U) = \min(\mathsf{ob}_!(?^{o_1}_{\kappa_1}.0), \mathsf{ob}_!(!^{o_2}_{\kappa_2}.0)) = \min(\infty, o_2) = o_2$
- $\mathsf{cap}_?(U) = \min(\mathsf{cap}_?(?^{o_1}_{\kappa_1}.0), \mathsf{cap}_?(!^{o_2}_{\kappa_2}.0)) = \min(\kappa_1, \infty) = \kappa_1$
- $\mathsf{ob}_?(U) = \min(\mathsf{ob}_?(?^{o_1}_{\kappa_1}.0), \mathsf{ob}_?(!^{o_2}_{\kappa_2}.0)) = \min(o_1, \infty) = o_1$
- $\mathsf{cap}_!(U) = \min(\mathsf{cap}_!(?^{o_1}_{\kappa_1}.0), \mathsf{cap}_!(!^{o_2}_{\kappa_2}.0)) = \min(\infty, \kappa_2) = \kappa_2$

This way, $\mathsf{con}_?(U)$ denotes $o_2 \leq \kappa_1$; similarly, $\mathsf{con}_!(U)$ denotes $o_1 \leq \kappa_2$. Then we have that $\mathsf{con}(U)$ holds when both $o_2 \leq \kappa_1$ and $o_1 \leq \kappa_2$ hold. This is indeed the condition needed for $\mathsf{rel}(U)$ to hold; notice that $U \to 0 \mid 0$ is the only reduction possible, and that $\mathsf{con}(0 \mid 0)$ trivially holds. Notice that letting $o_1 = \kappa_1 = o_2 = \kappa_2 = 0$ suffices for $\mathsf{rel}(U)$ to hold.

Having defined usages (and their associated notions), we move to define types.

Definition 3.8 *(Types with usages).* The syntax of types $\tau, \tau', \ldots$ builds upon usages as follows:

$$\begin{array}{rcll} \tau & ::= & \mathsf{chan}(\widetilde{\tau}\,;\, U) & \text{(channel types)} \\ & \mid & \langle l_i : \tau_i \rangle_{i \in I} & \text{(variant type)} \end{array}$$

Above, $\widetilde{\tau}$ indicates a sequence of types of length at most two. Type $\mathsf{chan}(\widetilde{\tau}\,;\, U)$ is associated to a channel that behaves according to usage U to exchange a tuple of values with types τ_1, τ_2. Notice that $\widetilde{\tau}$ can be empty; in that case, we write $\mathsf{chan}(-\,;\, U)$: a channel with this type behaves according to U without exchanging any values.

Differences with respect to the syntax of types in [38] are: (i) we do not consider Boolean nor product types, and (ii) we consider the *variant type* $\langle l_i : \tau_i \rangle_{i \in I}$ from [54] to denote the disjoint union of labelled types, where labels l_i ($i \in I$) are pairwise distinct. Variant types are essential to encode selection and branching in session types [17].

Typing contexts, ranged over $\Gamma, \Gamma', \ldots$, are produced by the following grammar:

$$\Gamma, \Gamma' ::= \emptyset \;\mid\; \Gamma, x : \tau$$

where '$\emptyset$' denotes the empty context. Given a context $\Gamma = x_1 : \tau_1, \cdots, x_n : \tau_n$, we write $\mathsf{dom}(\Gamma)$ to denote its domain, i.e., the set $\{x_1, \ldots, x_n\}$.

Following [38], we use '$\prec$' to denote a partial order that statically tracks the order in which channels are created. That is, $x \prec y$ means that x was created more recently than y. Typing judgements are indexed by '$\prec$'; they are of the form $\Gamma \vdash_\prec \widetilde{v}$ (for values) and $\Gamma \vdash_\prec P$ (for processes).

Before commenting on the typing rules, given in Fig. 6, we present some important auxiliary notions, extracted from [38].

Definition 3.9 *(Auxiliary operators on types).* We define the following auxiliary operators:

1. The unary operation $\uparrow^t$ on usages extends to types as follows:

$$\uparrow^t (\mathsf{chan}(\widetilde{\tau}\,;\, U)) = \mathsf{chan}(\widetilde{\tau}\,;\, \uparrow^t U)$$

2. The composition operation on types, denoted $\mid$, is defined as follows:

$$\mathsf{chan}(\widetilde{\tau}\,;\, U_1) \mid \mathsf{chan}(\widetilde{\tau}\,;\, U_2) \triangleq \mathsf{chan}(\widetilde{\tau}\,;\, (U_1 \mid U_2))$$
$$\langle l_i : \tau_i \rangle_{i \in I} \mid \langle l_i : \tau_i \rangle_{i \in I} \triangleq \langle l_i : \tau_i \rangle_{i \in I}$$

The generalisation of $\mid$ to typing contexts, denoted $(\Gamma_1 \mid \Gamma_2)(x)$, is defined as follows:

$$(\Gamma_1 \mid \Gamma_2)(x) = \begin{cases} \Gamma_1(x) \mid \Gamma_2(x) & \text{if } x \in \mathsf{dom}(\Gamma_1) \cap \mathsf{dom}(\Gamma_2) \\ \Gamma_1(x) & \text{if } x \in \mathsf{dom}(\Gamma_1) \setminus \mathsf{dom}(\Gamma_2) \\ \Gamma_2(x) & \text{if } x \in \mathsf{dom}(\Gamma_2) \setminus \mathsf{dom}(\Gamma_1) \end{cases}$$

3. The operator " $;_\prec$ " combines a type assignment $x : \mathsf{chan}(\widetilde{\tau}\,;\, \alpha^o_\kappa)$ and a context Γ into a new context. Precisely, $x : \mathsf{chan}(\widetilde{\tau}\,;\, \alpha^o_\kappa)\ ;_\prec \Gamma$ represents the context Γ', defined as follows:

$$\mathsf{dom}(\Gamma') \triangleq \{x\} \cup \mathsf{dom}(\Gamma)$$
$$\Gamma'(x) \triangleq \begin{cases} \mathsf{chan}(\widetilde{\tau}\,;\, \alpha^o_\kappa.U) & \text{if } \Gamma(x) = \mathsf{chan}(\widetilde{\tau}\,;\, U) \\ \mathsf{chan}(\widetilde{\tau}\,;\, \alpha^o_\kappa) & \text{if } x \notin \mathsf{dom}(\Gamma) \end{cases}$$
$$\Gamma'(y) \triangleq \begin{cases} \uparrow^\kappa \Gamma(y) & \text{if } y \neq x \wedge x \prec y \\ \uparrow^{\kappa+1} \Gamma(y) & \text{if } y \neq x \wedge x \not\prec y \end{cases}$$

$$\frac{}{x:\tau \vdash_{\prec} x:\tau}\ (\text{T}\pi\text{-Var}) \qquad \frac{\Gamma_1 \vdash_{\prec} v_1:\tau_1 \quad \Gamma_2 \vdash_{\prec} v_2:\tau_2 \quad \widetilde{v}=v_1,v_2 \ \ \widetilde{\tau}=\tau_1,\tau_2}{\Gamma_1 \mid \Gamma_2 \vdash_{\prec} \widetilde{v}:\widetilde{\tau}}\ (\text{T}\pi\text{-Tup})$$

$$\frac{\Gamma \vdash_{\prec} v:\tau_j \quad \exists j \in I}{\Gamma \vdash_{\prec} l_j\_v:\langle l_i:\tau_i\rangle_{i\in I}}\ (\text{T}\pi\text{-LVal}) \qquad \frac{}{\emptyset \vdash_{\prec} \mathbf{0}}\ (\text{T}\pi\text{-Nil})$$

$$\frac{\Gamma, x:\mathsf{chan}(\widetilde{\tau};\ U) \vdash_{\prec\cup\{(x,y)\,|\,y\in \mathrm{fn}(P)\setminus\{x\}\}} P \quad \mathsf{rel}(U)}{\Gamma \vdash_{\prec} (\boldsymbol{\nu} x)P}\ (\text{T}\pi\text{-Res}) \qquad \frac{\Gamma_1 \vdash_{\prec} P \quad \Gamma_2 \vdash_{\prec} Q}{\Gamma_1 \mid \Gamma_2 \vdash_{\prec} P \mid Q}\ (\text{T}\pi\text{-Par})$$

$$\frac{\Gamma_1 \vdash_{\prec} P \quad \Gamma_2 \vdash_{\prec} \widetilde{v}:\widetilde{\tau}}{x:\mathsf{chan}(\widetilde{\tau};\ !^0_\kappa)\ ;_{\prec}\ (\Gamma_1 \mid \Gamma_2) \vdash_{\prec} \overline{x}\langle\widetilde{v}\rangle.P}\ (\text{T}\pi\text{-Out}) \qquad \frac{\Gamma, \widetilde{y}:\widetilde{\tau} \vdash_{\prec} P}{x:\mathsf{chan}(\widetilde{\tau};\ ?^0_\kappa)\ ;_{\prec}\ \Gamma \vdash_{\prec} x(\widetilde{y}).P}\ (\text{T}\pi\text{-In})$$

$$\frac{\Gamma_1 \vdash_{\prec} v:\langle l_i:\tau_i\rangle_{i\in I} \qquad \Gamma_2, x_i:\tau_i \vdash_{\prec} P_i \quad \forall i\in I}{\Gamma_1 \mid \Gamma_2 \vdash_{\prec} \mathbf{case}\ v\ \mathbf{of}\ \{l_i\_x_i \triangleright P_i\}_{i\in I}}\ (\text{T}\pi\text{-Case})$$

Fig. 6. Typing rules for the π-calculus (§ 3.2.1). Rules for values appear in the first line; the remaining rules are for processes.

The typing rules for values and processes are given in Fig. 6. All rules are as in [38], except for the new rules (Tπ-LVal) and (Tπ-Case), which type a choice: the former types a variant value with a variant type; the latter types a case process using a variant value as its guard. We discuss the remaining rules. Rules (Tπ-Var) and (Tπ-Tup) are standard. Rule (Tπ-Nil) states that the terminated process is typed under the empty context. Rule (Tπ-Res) states that $(\boldsymbol{\nu} x)P$ is well-typed if the usage for x is reliable (cf. Definition 3.7). Rule (Tπ-Par) states that the parallel composition of processes P and Q (typable under Γ_1 and Γ_2, respectively) is well-typed under the composed typing context $\Gamma_1 \mid \Gamma_2$ (Definition 3.9(2)). Note that, unlike Rule (T-cut) in Fig. 5, P and Q need not to share any channels to be composed. Rules (Tπ-Out) and (Tπ-In) type output and input processes with dyadic communication in a typing context where the ' $;_{\prec}$ ' operator (Definition 3.9(3)) is used to increase the obligation level of the channels in continuation P.

3.2.3. Properties

Type soundness of the type system given in Fig. 6 implies that well-typed processes are deadlock-free (cf. Definition 3.2). We now state these technical results from [38] and discuss changes in their proofs, which require minimal modifications.

First, typing is preserved by the structural relation $\preceq$ (cf. Definition 3.1):

Lemma 3.1. *If $\Gamma \vdash_{\prec} P$ and $P \preceq Q$ then $\Gamma \vdash_{\prec} Q$.*

The proof of type preservation in [38] relies on other auxiliary results (such as substitution), which we do not recall here. To state the type preservation result, we need the following auxiliary definition:

Definition 3.10 *(Context reduction).* We write $\Gamma \to \Gamma'$ when one of the following holds:

1. $\Gamma = \Gamma_1, x:\mathsf{chan}(\widetilde{\tau};\ U)$ and $\Gamma' = \Gamma_1, x:\mathsf{chan}(\widetilde{\tau};\ U')$ with $U \to U'$ (cf. Definition 3.6), for some Γ_1, x, $\widetilde{\tau}$, U and U'.
2. $\Gamma = \Gamma_1, x:\langle l_i:\tau_i\rangle_{i\in I}$ and $\Gamma' = \Gamma_1, x:\tau_j$, with $j \in I$, for some Γ_1 and x.

Above, Item (1) is as in [38]. Item (2) is required for characterizing reductions of the case construct. We now state type preservation.

Theorem 3.3 *(Type preservation). If $\Gamma \vdash_{\prec} P$ and $P \to Q$, then $\Gamma' \vdash_{\prec} Q$ for some Γ' such that $\Gamma' = \Gamma$ or $\Gamma \to \Gamma'$.*

The proof is by induction on the derivation $P \to Q$, following closely the proof in [38]. An additional case is needed due to the reduction rule of the case construct (Rule (Rπ-Case)):

$$\mathbf{case}\ l_j\_v\ \mathbf{of}\ \{l_i\_x_i \triangleright P_i\}_{i\in I} \to P_j[v/x_j] \quad (j \in I)$$

As a result of this reduction, we have $\Gamma \to \Gamma'$ because of Definition 3.10(2).

The following important result extends Theorem 2 in [38] with the case construct:

Theorem 3.4 *(Deadlock freedom). If $\emptyset \vdash_{\prec} P$ and one of the following holds*

$$
\begin{aligned}
[\![\mathbf{0}]\!]^f_{\mathsf{u}} &\triangleq \mathbf{0}\\
[\![(\boldsymbol{\nu} xy)P]\!]^f_{\mathsf{u}} &\triangleq (\boldsymbol{\nu} c)[\![P]\!]^{f,\{x,y\mapsto c\}}_{\mathsf{u}}\\
[\![P \mid Q]\!]^f_{\mathsf{u}} &\triangleq [\![P]\!]^f_{\mathsf{u}} \mid [\![Q]\!]^f_{\mathsf{u}}\\
[\![\overline{x}\langle v\rangle.P]\!]^f_{\mathsf{u}} &\triangleq (\boldsymbol{\nu} c)\overline{f_x}\langle f_v, c\rangle.[\![P]\!]^{f,\{x\mapsto c\}}_{\mathsf{u}}\\
[\![x(y).P]\!]^f_{\mathsf{u}} &\triangleq f_x(y,c).[\![P]\!]^{f,\{x\mapsto c\}}_{\mathsf{u}}\\
[\![x \lhd l_j.P]\!]^f_{\mathsf{u}} &\triangleq (\boldsymbol{\nu} c)\overline{f_x}\langle l_j\_c\rangle.[\![P]\!]^{f,\{x\mapsto c\}}_{\mathsf{u}}\\
[\![x \rhd \{l_i : P_i\}_{i\in I}]\!]^f_{\mathsf{u}} &\triangleq f_x(y).\ \mathbf{case}\ y\ \mathbf{of}\ \{l_i\_c \rhd [\![P_i]\!]^{f,\{x\mapsto c\}}_{\mathsf{u}}\}_{i\in I}
\end{aligned}
$$

Fig. 7. Encoding session π-calculus processes into the dyadic π-calculus, under a renaming function f on names/channels.

$$
\begin{aligned}
[\![\mathbf{end}]\!]_{\mathsf{u}} &= \mathsf{chan}(-\,;\,0)\\
[\![?T.S]\!]_{\mathsf{u}} &= \mathsf{chan}([\![T]\!]_{\mathsf{u}}, [\![S]\!]_{\mathsf{u}}\,;\,?^0_0)\\
[\![!T.S]\!]_{\mathsf{u}} &= \mathsf{chan}([\![T]\!]_{\mathsf{u}}, [\![\overline{S}]\!]_{\mathsf{u}}\,;\,!^0_0)\\
[\![\&\{l_i : S_i\}_{i\in I}]\!]_{\mathsf{u}} &= \mathsf{chan}(\langle l_i : [\![S_i]\!]_{\mathsf{u}}\rangle_{i\in I}\,;\,?^0_0)\\
[\![\oplus\{l_i : S_i\}_{i\in I}]\!]_{\mathsf{u}} &= \mathsf{chan}(\langle l_i : [\![\overline{S_i}]\!]_{\mathsf{u}}\rangle_{i\in I}\,;\,!^0_0)
\end{aligned}
$$

Fig. 8. Encoding session types into usage types.

- $P \preceq (\boldsymbol{\nu}\widetilde{x})(\overline{x}\langle\widetilde{v}\rangle.Q_1 \mid Q_2)$
- $P \preceq (\boldsymbol{\nu}\widetilde{x})(x(\widetilde{z}).Q_1 \mid Q_2)$
- $P \preceq (\boldsymbol{\nu}\widetilde{x})(\mathbf{case}\, l_j\_v\ \mathbf{of}\ \{l_i\_x_i \rhd P_i\}_{i\in I} \mid Q)$

then $P \to R$, for some process R.

In [38], the proof of this theorem relies on a notion of *normal form* in which input and output prefixes act as guards for if-expressions and let-expressions (not present in our syntax); our case construct can be easily accommodated in such normal forms. The proof in [38] also uses: (i) an extension of $\preceq$ for replicated processes, (ii) mechanisms for uniquely identifying bound names, and (iii) an ordering on processes. In our case, we do not need (i) and can re-use (ii) and (iii) as they are. With these elements, the proof argument proceeds as in [38]. We finally have:

Corollary 3.1. *If $\emptyset \vdash_{\prec} P$ then P is deadlock-free, in the sense of Definition 3.2.*

It is worth noticing how both Theorem 3.2 and Theorem 3.4 have similar formulations: both properties state that processes can always reduce if they are well-typed (under the empty typing context) and have an appropriate structure (cf., condition $live(P)$ in Theorem 3.2 and condition $P \preceq (\boldsymbol{\nu}\widetilde{x})(x(\widetilde{z}).Q \mid R)$ or $P \preceq (\boldsymbol{\nu}\widetilde{x})(\overline{x}\langle\widetilde{v}\rangle.Q \mid R)$ in Theorem 3.4).

3.2.4. On deadlock freedom by encoding

We use encodings to relate classes of (typed) processes induced by the type systems given so far. To translate a session typed process into a usage typed process, we follow the encoding suggested in [39] and developed in [17], which mimics the sequential structure of a session by sending its continuation as a payload over a channel. This continuation-passing style encoding of processes is denoted $[\![\cdot]\!]^f_{\mathsf{u}}$, where f is a renaming function from channels to fresh names; see Fig. 7. We write f_x to stand for $f(x)$ and $f, \{x \mapsto c\}$ to denote that the entry for x in the renaming f is updated to c. We write $f, \{x, y \mapsto c\}$ to mean that both x and y are updated to c.

We also need to formally relate session types to usage types. To this end, we define $[\![\cdot]\!]_{\mathsf{u}}$ in Fig. 8. Two points are noteworthy: (i) it suffices to generate a usage type with obligations and capabilities equal to 0, and (ii) only sequential usages are needed to encode session types. We now extend this encoding from types to typing contexts:

Definition 3.11. Given $[\![\cdot]\!]_{\mathsf{u}}$ as in Fig. 8, and with a slight abuse of notation, we write $[\![\cdot]\!]^f_{\mathsf{u}}$ to denote the encoding of session type contexts Γ into usage typing contexts that is inductively defined as follows:

$$[\![\emptyset]\!]^f_{\mathsf{u}} \triangleq \emptyset \qquad [\![\Gamma, x : T]\!]^f_{\mathsf{u}} \triangleq [\![\Gamma]\!]^f_{\mathsf{u}}, f_x : [\![T]\!]_{\mathsf{u}}$$

$$[\![\overline{x}\langle y\rangle.P]\!]_\ell \triangleq \overline{x}(z).([z\leftrightarrow y] \mid [\![P]\!]_\ell)$$
$$[\![(\boldsymbol{\nu} xy)P]\!]_\ell \triangleq (\boldsymbol{\nu} w)[\![P]\!]_\ell[w/x][w/y] \quad w \notin \mathsf{fn}(P)$$

Fig. 9. Encoding session π-calculus processes into the linear logic processes (cf. § 3.1). It is defined as a homomorphism for the other process constructs.

The next results relate deadlock freedom, typing and the encodings in [17], thus formalising the indirect approach to deadlock freedom.

Proposition 3.1. *Let P be a deadlock-free session process. Then $[\![P]\!]^f_u$ is a deadlock-free π-process.*

Proof. The encoding of terms given in Fig. 7 preserves the nature of the prefixes in P (outputs are directly translated as outputs, and similarly for inputs) and is defined homomorphically with respect to parallel composition. Then, it is easy to see that if P reduces then $[\![P]\!]^f_u$ can immediately match that reduction; thanks to the definitions of deadlock freedom in each language (Definition 2.3 and Definition 3.2, respectively), this suffices to conclude the thesis. □

The following result states deadlock freedom by encodability, following [8].

Corollary 3.2. *Let $\vdash_{\mathtt{ST}} P$ be a session process. If $\vdash_{\prec} [\![P]\!]^f_u$ is deadlock-free then P is deadlock-free.*

4. Separating classes of deadlock-free typed processes

Up to here, we have summarized three existing type systems for the π-calculus:

- Session types, with judgement $\Gamma \vdash_{\mathtt{ST}} P$ (§ 2.2);
- Session types based on linear logic, with judgement $P \vdash_{\mathtt{LL}} \Delta$ (§ 3.1);
- Usage types, with judgement $\Gamma \vdash_{\prec} P$ (§ 3.2.2).

Here we establish formal relationships between these type systems. As already mentioned, our approach consists in defining $\mathcal{L}$ and $\mathcal{K}$, the class of deadlock-free session-typed processes induced by the type systems in § 3.1 and § 3.2.2, respectively. Our main result is that $\mathcal{L} \subset \mathcal{K}$ (Corollary 4.1), a strict inclusion that *separates* these two classes of processes. To obtain this separation result, we define $\mu\mathcal{K}$, a strict sub-class of $\mathcal{K}$ which we show to coincide with $\mathcal{L}$.

4.1. Classes of deadlock-free processes

4.1.1. The classes $\mathcal{L}$ and $\mathcal{K}$

We start by defining classes $\mathcal{L}$ and $\mathcal{K}$, for which we require some auxiliary definitions. The following encoding addresses minor syntactic differences between session typed processes (cf. § 2) and the processes typable in the linear logic interpretation of session types (cf. § 3.1). Such differences concern free output actions and the double restriction operator:

Definition 4.1. Let P be a session process. The auxiliary encoding $[\![\cdot]\!]_\ell$ from the session processes (cf. § 2) into the linear logic processes (cf. § 3.1) is defined in Fig. 9.

We also need an encoding of session types into linear logic propositions. The encoding, also denoted $[\![\cdot]\!]_\ell$ and given in Fig. 10, simply follows the essence of the linear logic interpretation. We extend it to typing contexts as follows:

Definition 4.2. The encoding $[\![\cdot]\!]_\ell$ of session type contexts Γ into linear logic typing contexts is defined as:

$$[\![\emptyset]\!]_\ell \triangleq \emptyset \qquad [\![\Gamma, x:T]\!]_\ell \triangleq [\![\Gamma]\!]_\ell, x:[\![T]\!]_\ell$$

It is not difficult to see that the encoding in Definition 4.1 is operationally correct. That is, $[\![\cdot]\!]_\ell$ preserves and reflects reductions of session processes. It also preserves the encoding of types in Fig. 10 and Definition 4.2.

The next definition allows us to abstract away from differences in the way type systems handle process **0**: in the usages type system, process **0** can only be typed under the empty typing context, whereas in the session type systems the typing context can be non-empty (subject to conditions).

Definition 4.3 *(Core context).* Given $\Gamma \vdash_{\mathtt{ST}} P$, we write $\Gamma^{\downarrow}$ to denote the *core context* of Γ with respect to P. Context $\Gamma^{\downarrow}$ is defined so that $\Gamma = \Gamma^{\downarrow}, \Gamma'$ holds for some Γ', satisfying the following conditions: (i) $\mathsf{un}(\Gamma')$; (ii) $x:\mathbf{end} \in \Gamma'$ implies $x \notin \mathsf{fn}(P)$; and (ii) $x:T \in \Gamma^{\downarrow}$ implies $x \in \mathsf{fn}(P)$.

$$
\begin{aligned}
[\![\mathbf{end}]\!]_\ell &= \bullet \\
[\![?T.S]\!]_\ell &= [\![T]\!]_\ell \parr [\![S]\!]_\ell \\
[\![!T.S]\!]_\ell &= [\![\overline{T}]\!]_\ell \otimes [\![S]\!]_\ell \\
[\![\&\{l_i : S_i\}_{i\in I}]\!]_\ell &= \&\{l_i : [\![S_i]\!]_\ell\}_{i\in I} \\
[\![\oplus\{l_i : S_i\}_{i\in I}]\!]_\ell &= \oplus\{l_i : [\![S_i]\!]_\ell\}_{i\in I}
\end{aligned}
$$

Fig. 10. Encoding session types into linear logic types.

Notice that, by Theorem 2.1 (strengthening), $\Gamma \vdash_{\mathrm{ST}} P$ implies $\Gamma^{\downarrow} \vdash_{\mathrm{ST}} P$.

Recall that we write $\mathbb{P}$ to denote the class of session π-calculus processes generated by the grammar in Fig. 2. Formally, $\mathcal{L}$ and $\mathcal{K}$ are classes of processes in $\mathbb{P}$, defined below.

Definition 4.4 *($\mathcal{L}$ and $\mathcal{K}$).* The classes $\mathcal{L}$ and $\mathcal{K}$ are defined as follows:

$$
\begin{aligned}
\mathcal{L} &\triangleq \left\{ P \in \mathbb{P} \,:\, \exists \Gamma.\, (\Gamma \vdash_{\mathrm{ST}} P \wedge [\![P]\!]_\ell \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell) \right\} \\
\mathcal{K} &\triangleq \left\{ P \in \mathbb{P} \,:\, \exists \Gamma, f.\, (\Gamma \vdash_{\mathrm{ST}} P \wedge [\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f} \vdash_{\prec} [\![P]\!]_{\mathsf{u}}^{f}) \right\}
\end{aligned}
$$

In words, $\mathcal{L}$ contains those well-typed session π-calculus processes whose corresponding translation as a process in § 3.1 (using $[\![\cdot]\!]_\ell$ as in Definition 4.1) is also typable in the linear logic interpretation, with propositions obtained using the encoding on types $[\![\cdot]\!]_\ell$ (Fig. 10). Similarly, $\mathcal{K}$ contains those well-typed session π-calculus processes whose corresponding translation as a dyadic π-calculus process (using $[\![\cdot]\!]_{\mathsf{u}}^{f}$ as in Fig. 7) is also typable under Kobayashi's type system, with usage types derived using the encoding on types $[\![\cdot]\!]_{\mathsf{u}}$ (Fig. 8).

Notice that $\mathcal{L}$ and $\mathcal{K}$ contain both composable and uncomposable processes (cf. Definition 2.2). As informally discussed in the Introduction, processes in $\mathcal{L}$ and $\mathcal{K}$ satisfy the *progress* property, defined in [7] and further studied in [8]. As a consequence:

- *Uncomposable processes* in $\mathcal{L}$ (typable with $\Delta = \cdot$) are deadlock-free by following Theorem 3.2. Similarly, *uncomposable processes* in $\mathcal{K}$ (typable with $\Gamma = \emptyset$) are deadlock-free by the indirect approach formalised by Theorem 3.4 and Definition 3.2 and Corollary 3.2.
- *Composable processes* in $\mathcal{L}$ (typable with $\Delta \neq \cdot$) and $\mathcal{K}$ (typable with $\Gamma \neq \emptyset$) may be stuck, because they lack communicating counterparts as described by their (non-empty) typing context. These missing counterparts will be formalized as a *catalyzer* [7] that allows a process to further reduce, thereby "unstucking it".

Although we are interested in the (sub)class of processes that satisfy deadlock freedom, we have defined $\mathcal{L}$ and $\mathcal{K}$ more generally as processes in $\mathbb{P}$ satisfying progress; this simplifies the definition and presentation of our technical contributions.

4.1.2. The class $\mu\mathcal{K}$

We now define $\mu\mathcal{K}$, a strict class of $\mathcal{K}$ obtained by internalizing key features of linear logic interpretation in § 3.1 into the rules of Fig. 6. Considering the type system is summarized in § 3.2, $\mu\mathcal{K}$ will arise from a type system modified as follows:

Types Because session types are strictly sequential, we restrict Definition 3.8 to channel types of the form $\mathsf{chan}(\tau_1, \tau_2 ; U)$ where U is a *sequential usage* (cf. Definition 3.3). All other notions (obligations and capabilities, the semantics of usages, the notion of reliability) are kept unchanged.

Typing Rules The type system considers judgements of the form $\Gamma \vdash^{\mu}_{\prec} P$. The typing rules are given in Fig. 11; they are based on those in Fig. 6 with the following modifications. First, we consider Rule (Tπ-BOut), which accounts for bound output as used in the type discipline in § 3.1. Second, to account for the "composition plus hiding" principle, we merge Rules (Tπ-Res) and (Tπ-Par) into Rule (Tπ-Par+Res). In this modified rule, the parallel usage appears exclusively for the purpose of ensuring reliability. Finally, we include Rule (Tπ-IndPar) to account for independent parallel composition as expressed by the mix principle in § 3.1.

We use this modified type system to define $\mu\mathcal{K}$, following Definition 4.4:

Definition 4.5 *($\mu\mathcal{K}$).* The class of processes $\mu\mathcal{K}$ is defined as follows:

$$
\mu\mathcal{K} \triangleq \left\{ P \in \mathbb{P} \,:\, \exists \Gamma, f.\, (\Gamma \vdash_{\mathrm{ST}} P \wedge [\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f} \vdash^{\mu}_{\prec} [\![P]\!]_{\mathsf{u}}^{f}) \right\}
$$

$$\text{(T}\pi\text{-Var)}\ \frac{}{x:\tau \vdash^{\mu}_{\prec} x:\tau} \qquad \text{(T}\pi\text{-Tup)}\ \frac{\Gamma_1 \vdash^{\mu}_{\prec} v_1:\tau_1 \qquad \Gamma_2 \vdash^{\mu}_{\prec} v_2:\tau_2 \qquad \widetilde{v}=v_1,v_2\ \ \widetilde{\tau}=\tau_1,\tau_2}{\Gamma_1 \mid \Gamma_2 \vdash^{\mu}_{\prec} \widetilde{v}:\widetilde{\tau}} \qquad \text{(T}\pi\text{-LVal)}\ \frac{\Gamma \vdash^{\mu}_{\prec} v:\tau_j \quad \exists j\in I}{\Gamma \vdash^{\mu}_{\prec} l_j\_v:\langle l_i:\tau_i\rangle_{i\in I}}$$

$$\text{(T}\pi\text{-Nil)}\ \frac{}{\emptyset \vdash^{\mu}_{\prec} \mathbf{0}} \qquad \text{(T}\pi\text{-IndPar)}\ \frac{\Gamma_1 \vdash^{\mu}_{\prec} P \qquad \Gamma_2 \vdash^{\mu}_{\prec} Q \qquad \mathsf{dom}(\Gamma_1)\cap\mathsf{dom}(\Gamma_2)=\emptyset}{\Gamma_1,\Gamma_2 \vdash^{\mu}_{\prec} P \mid Q}$$

$$\text{(T}\pi\text{-Par+Res)}\ \frac{\begin{array}{c}\Gamma_1, x:\mathsf{chan}(\tau\,;\,U_1) \vdash^{\mu}_{\prec\cup\prec_1} P_1 \qquad \Gamma_2, x:\mathsf{chan}(\tau\,;\,U_2) \vdash^{\mu}_{\prec\cup\prec_2} P_2 \qquad \mathsf{dom}(\Gamma_1)\cap\mathsf{dom}(\Gamma_2)=\emptyset \\ i\in\{1,2\} \qquad \prec_i=\{(x,y)\mid y\in \mathsf{fn}(P_i)\setminus\{x\}\} \qquad \mathsf{rel}(U_1 \mid U_2)\end{array}}{\Gamma_1 \mid \Gamma_2 \vdash^{\mu}_{\prec} (\boldsymbol{\nu} x)(P_1 \mid P_2)}$$

$$\text{(T}\pi\text{-BOut)}\ \frac{\begin{array}{c}\Gamma_1, y:\mathsf{chan}(\tau\,;\,U_1) \vdash^{\mu}_{\prec'} y:\mathsf{chan}(\tau\,;\,U_1), y':\mathsf{chan}(\tau'\,;\,U) \qquad \Gamma_2, y:\mathsf{chan}(\tau\,;\,U_2) \vdash^{\mu}_{\prec'} P \\ \mathsf{rel}(U_1 \mid U_2) \qquad \prec'=\prec\cup\{(y,z)\mid z\in\mathsf{fn}(P)\setminus\{y\}\} \qquad \widetilde{y}=y,y'\ \ \widetilde{\tau}=\tau,\tau'\end{array}}{x:\mathsf{chan}(\widetilde{\tau}\,;\,!^0_\kappa)\,;_\prec (\Gamma_1 \mid \Gamma_2) \vdash^{\mu}_{\prec} (\boldsymbol{\nu} y)\overline{x}\langle\widetilde{y}\rangle.P}$$

$$\text{(T}\pi\text{-In)}\ \frac{\Gamma, \widetilde{z}:\widetilde{\tau} \vdash^{\mu}_{\prec} P}{x:\mathsf{chan}(\widetilde{\tau}\,;\,?^0_\kappa)\,;_\prec \Gamma \vdash^{\mu}_{\prec} x(\widetilde{z}).P} \qquad \text{(T}\pi\text{-Case)}\ \frac{\Gamma_1 \vdash^{\mu}_{\prec} v:\langle l_i:\tau_i\rangle_{i\in I} \qquad \Gamma_2, x_i:\tau_i \vdash^{\mu}_{\prec} P_i \quad \forall i\in I}{\Gamma_1 \mid \Gamma_2 \vdash^{\mu}_{\prec} \mathbf{case}\ v\ \mathbf{of}\ \{l_i\_x_i \triangleright P_i\}_{i\in I}}$$

Fig. 11. Typing rules for the π-calculus (§ 3.2.1), which specialize those in Fig. 6 to define the class $\mu\mathcal{K}$. Typing rules for values appear in the first line; the remaining typing rules are for processes.

Considerations about (un)composable processes in $\mu\mathcal{K}$ hold exactly as described for $\mathcal{K}$.

4.2. Main results

We start by separating $\mathcal{K}$ and $\mu\mathcal{K}$. First, we have the following:

Lemma 4.1. *Let P be a dyadic π-calculus process as in § 3.2.1. If $\Gamma \vdash^{\mu}_{\prec} P$ then $\Gamma \vdash_{\prec} P$.*

Proof. By induction on the type derivation for P, with a case analysis on the last applied typing rule. We rely on the analogue property for values (if $\Gamma \vdash^{\mu}_{\prec} v$ then $\Gamma \vdash_{\prec} v$), which is immediate. We discuss only the cases featuring key differences between $\vdash^{\mu}_{\prec}$ (Fig. 11) and $\vdash_{\prec}$ (Fig. 6) arise:

- If the last applied rule is (Tπ-BOut), then $P=(\boldsymbol{\nu} y)\overline{x}\langle y,y'\rangle.P'$ and

$$\frac{\begin{array}{c}\Gamma_1, y:\mathsf{chan}(\tau\,;\,U_1) \vdash^{\mu}_{\prec'} y:\mathsf{chan}(\tau\,;\,U_1), y':\mathsf{chan}(\tau'\,;\,U) \qquad \Gamma_2, y:\mathsf{chan}(\tau\,;\,U_2) \vdash^{\mu}_{\prec'} P \\ \mathsf{rel}(U_1 \mid U_2) \qquad \prec'=\prec\cup\{(y,z)\mid z\in\mathsf{fn}(P)\setminus\{y\}\}\end{array}}{\Gamma \vdash^{\mu}_{\prec} (\boldsymbol{\nu} y)\overline{x}\langle y,y'\rangle.P}$$

  with $\Gamma = x:\mathsf{chan}(\tau,\tau'\,;\,!^0_\kappa)\,;_\prec (\Gamma_1 \mid \Gamma_2)$. We have $\Gamma_1, y:\mathsf{chan}(\tau\,;\,U_1) \vdash_{\prec'} y:\mathsf{chan}(\tau\,;\,U_1), y':\mathsf{chan}(\tau'\,;\,U)$. Also, by IH: $\Gamma_2, y:\mathsf{chan}(\tau\,;\,U_2) \vdash_{\prec'} P$.
  We show that P can be typed using Rules (Tπ-Out) and (Tπ-Res) in sequence. First we have:

$$\frac{\Gamma_1, y:\mathsf{chan}(\tau\,;\,U_1) \vdash_{\prec'} y:\mathsf{chan}(\tau\,;\,U_1), y':\mathsf{chan}(\tau'\,;\,U) \qquad \Gamma_2, y:\mathsf{chan}(\tau\,;\,U_2) \vdash_{\prec'} P}{x:\mathsf{chan}(\tau,\tau'\,;\,!^0_\kappa)\,;_\prec (\Gamma_1 \mid \Gamma_2 \mid y:\mathsf{chan}(\tau\,;\,U_1 \mid U_2)) \vdash_{\prec} \overline{x}\langle y,y'\rangle.P}$$

  Now, because by assumption we have $\mathsf{rel}(U_1 \mid U_2)$, we can use Rule (T$\pi$-Res) to derive

$$\frac{x:\mathsf{chan}(\tau,\tau'\,;\,!^0_\kappa)\,;_\prec (\Gamma_1 \mid \Gamma_2 \mid y:\mathsf{chan}(\tau\,;\,U_1 \mid U_2)) \vdash_{\prec} \overline{x}\langle y,y'\rangle.P \qquad \mathsf{rel}(U_1 \mid U_2)}{x:\mathsf{chan}(\tau,\tau'\,;\,!^0_\kappa)\,;_\prec (\Gamma_1 \mid \Gamma_2) \vdash^{\mu}_{\prec} (\boldsymbol{\nu} y)\overline{x}\langle y,y'\rangle.P}$$

- If the last applied rule is (Tπ-Par+Res), then $P = (\boldsymbol{\nu} x)(P_1 \mid P_2)$ and

$$\frac{\Gamma_1, x : \mathsf{chan}(\tau\,;\, U_1) \vdash^{\mu}_{\prec\cup\prec_1} P_1 \qquad \Gamma_2, x : \mathsf{chan}(\tau\,;\, U_2) \vdash^{\mu}_{\prec\cup\prec_2} P_2 \qquad \mathsf{dom}(\Gamma_1) \cap \mathsf{dom}(\Gamma_2) = \emptyset \\ i \in \{1,2\} \qquad \prec_i = \{(x,y) \mid y \in \mathsf{fn}(P_i) \setminus \{x\}\} \qquad \mathsf{rel}(U_1 \mid U_2)}{\Gamma \vdash^{\mu}_{\prec} (\boldsymbol{\nu} x)(P_1 \mid P_2)}$$

  where $\Gamma = \Gamma_1 \mid \Gamma_2$. By IH, we have $\Gamma_i, x : \mathsf{chan}(\tau\,;\, U_i) \vdash_{\prec\cup\prec_i} P_i$, with $i \in \{1,2\}$.
  We show that P can be typed using Rules (Tπ-Par) and (Tπ-Res) in sequence. First, we have:

$$\frac{\Gamma_1, x : \mathsf{chan}(\tau\,;\, U_1) \vdash_{\prec\cup\prec_1\cup\prec_2} P_1 \qquad \Gamma_2, x : \mathsf{chan}(\tau\,;\, U_2) \vdash_{\prec\cup\prec_1\cup\prec_2} P_2}{\Gamma_1 \mid \Gamma_2 \mid x : \mathsf{chan}(\tau\,;\, U_1 \mid U_2) \vdash_{\prec\cup\prec_1\cup\prec_2} P_1 \mid P_2}$$

  where we have weakened, in both cases, the partial orders for P_1 and P_2. Now, because by assumption we have $\mathsf{rel}(U_1 \mid U_2)$, we can use Rule (Tπ-Res) to derive

$$\frac{\Gamma_1 \mid \Gamma_2 \mid x : \mathsf{chan}(\tau\,;\, U_1 \mid U_2) \vdash_{\prec\cup\prec_1\cup\prec_2} P_1 \mid P_2 \qquad \mathsf{rel}(U_1 \mid U_2)}{\Gamma_1 \mid \Gamma_2 \vdash_{\prec} (\boldsymbol{\nu} x)(P_1 \mid P_2)}$$

- Finally, if the last applied rule is (Tπ-IndPar), then process $P = P_1 \mid P_2$, which can be immediately typed using Rule (Tπ-Par). □

As a result of Lemma 4.1, properties derived from typing in § 3.2 (type preservation and deadlock freedom) hold in the modified type system $\vdash^{\mu}_{\prec}$ and apply to processes in $\mu\mathcal{K}$. Thus, $\mu\mathcal{K} \subseteq \mathcal{K}$.

The modifications defined in § 4.1.2 make $\vdash^{\mu}_{\prec}$ a strict subsystem of the system $\vdash_{\prec}$ in § 3.2. Indeed, because the converse of Lemma 4.1 does not hold, there are processes in $\mathcal{K}$ but not in $\mu\mathcal{K}$:

Lemma 4.2. $\mu\mathcal{K} \subset \mathcal{K}$.

Proof. Because $\mu\mathcal{K}$ is obtained by restricting the typing rules of $\mathcal{K}$, it is immediate that $\mu\mathcal{K} \subseteq \mathcal{K}$. In the following, we show that this inclusion is strict, by showing that $\mathcal{K}$ contains (deadlock-free) session processes not in $\mu\mathcal{K}$. A representative example is:

$$P \triangleq (\boldsymbol{\nu} a_1 b_1)(\boldsymbol{\nu} a_2 b_2)(a_1(x).\, \overline{a_2}\langle x\rangle \mid \overline{b_1}\langle \mathbf{n}\rangle.\, b_2(z))$$
$$[\![P]\!]^{f}_{\mathsf{u}} = (\boldsymbol{\nu} c)(\boldsymbol{\nu} d)(c(x,w).\, (\boldsymbol{\nu} d')\overline{d}\langle x, d'\rangle \mid (\boldsymbol{\nu} c')\overline{c}\langle \mathbf{n}, c'\rangle.\, d(z,u))$$

This process is not in $\mu\mathcal{K}$ because $[\![P]\!]^{f}_{\mathsf{u}}$ involves the composition of two parallel processes which share two sessions. Hence, $[\![P]\!]^{f}_{\mathsf{u}}$ is typable in $\vdash_{\prec}$ (using Rules (Tπ-Par) and (Tπ-Res) in Fig. 6) but not in $\vdash^{\mu}_{\prec}$, because of the forms of composition admitted by Rule (Tπ-Par+Res) in Fig. 11. □

We now show that $\mathcal{L}$ and $\mu\mathcal{K}$ coincide. We need an important property, given by Lemma 4.4, which connects our encodings of (dual) session types into usage types with reliability (Definition 3.7), a central notion to the type systems for deadlock freedom in Fig. 6 and Fig. 11. First we have the following result, which connects our encodings of types and the notion of duality.

Lemma 4.3. *Let T, S be session types. Then, the following hold:*
(i) $\overline{T} = S$ if and only if $[\![\overline{T}]\!]_{\ell} = [\![S]\!]_{\ell}$; (ii) $\overline{T} = S$ if and only if $[\![\overline{T}]\!]_{\mathsf{u}} = [\![S]\!]_{\mathsf{u}}$.

Proof. By induction on the duality relation of session types. □

Given a channel type $\tau = \mathsf{chan}(\widetilde{\tau}\,;\, U)$, we write $u(\tau)$ to denote the usage U.

Lemma 4.4. *Let T be a session type. Then* $\mathsf{rel}(u([\![T]\!]_{\mathsf{u}}) \mid u([\![\overline{T}]\!]_{\mathsf{u}}))$ *holds.*

Proof. The proof proceeds by induction on the structure of T, using Lemma 4.3.

- $T = \mathbf{end}$. By duality, $\overline{T} = \mathbf{end}$. Then $[\![T]\!]_{\mathsf{u}} = [\![\overline{T}]\!]_{\mathsf{u}} = \mathsf{chan}(-\,;\, 0)$. Thus, $u([\![T]\!]_{\mathsf{u}}) \mid u([\![\overline{T}]\!]_{\mathsf{u}}) = 0 \mid 0$. Notice that $0 \mid 0 \not\to$ and that $\mathsf{con}(0 \mid 0)$ holds trivially. Therefore, by Definition 3.7, $\mathsf{rel}(0 \mid 0)$.

- $T = !T_1.T_2$ for some T_1, T_2. By definition of duality, $\overline{T} = ?T_1.\overline{T_2}$. Then,

$$[\![T]\!]_u = \mathsf{chan}([\![T_1]\!]_u, [\![\overline{T_2}]\!]_u\,;\, !_0^0)$$
$$[\![\overline{T}]\!]_u = \mathsf{chan}([\![T_1]\!]_u, [\![\overline{T_2}]\!]_u\,;\, ?_0^0)$$

  Thus, $u([\![T]\!]_u) \mid u([\![\overline{T}]\!]_u) = !_0^0 \mid ?_0^0$. Notice that $!_0^0 \mid ?_0^0 \to 0 \mid 0 \not\to$. We examine $\mathsf{con}(\cdot)$ for both usages. First, $\mathsf{con}(!_0^0 \mid ?_0^0)$ holds because $0 \leq 0$; then, $\mathsf{con}(0 \mid 0)$ trivially holds. Therefore, by Definition 3.7, $\mathsf{rel}(!_0^0 \mid ?_0^0)$ holds.
- $T = ?T_1.T_2$, for some T_1, T_2. By definition of duality, $\overline{T} = !T_1.\overline{T_2}$. Then,

$$[\![T]\!]_u = \mathsf{chan}([\![T_1]\!]_u, [\![T_2]\!]_u\,;\, ?_0^0)$$
$$[\![\overline{T}]\!]_u = \mathsf{chan}([\![T_1]\!]_u, [\![T_2]\!]_u\,;\, !_0^0)$$

  Thus, $u([\![T]\!]_u) \mid u([\![\overline{T}]\!]_u) = ?_0^0 \mid !_0^0$ and the thesis holds as in the previous case.
- $T = \&\{l_i : S_i\}_{i\in I}$, for some S_i. By definition of duality, $\overline{T} = \oplus\{l_i : \overline{S_i}\}_{i\in I}$. Then,

$$[\![T]\!]_u = \mathsf{chan}(\langle l_i : [\![S_i]\!]_u\rangle_{i\in I}\,;\, ?_0^0)$$
$$[\![\overline{T}]\!]_u = \mathsf{chan}(\langle l_i : [\![S_i]\!]_u\rangle_{i\in I}\,;\, !_0^0)$$

  Thus, $u([\![T]\!]_u) \mid u([\![\overline{T}]\!]_u) = ?_0^0 \mid !_0^0$ and the thesis holds just as in case $T = ?T_1.T_2$.
- $T = \oplus\{l_i : S_i\}_{i\in I}$, for some S_i and $i \in I$: similar to the previous cases. □

We then have the following result:

Theorem 4.1. $\mathcal{L} = \mu\mathcal{K}$.

Proof (sketch). We prove two lemmas: (i) If $P \in \mathcal{L}$ then $P \in \mu\mathcal{K}$ and (ii) If $P \in \mu\mathcal{K}$ then $P \in \mathcal{L}$. Both lemmas are proven by structural induction on P; see Appendix A.1 (Page 28) for details. □

Therefore, we have the following corollary, which attests that the class of deadlock-free session processes induced by linear logic interpretations of session types (cf. § 3.1) is strictly included in the class induced by the indirect approach of [17] (cf. § 3.2).

Corollary 4.1. $\mathcal{L} \subset \mathcal{K}$.

The fact that (deadlock-free) processes such as P (cf. Lemma 4.2) are not included in $\mathcal{L}$ is informally discussed by Caires et al. in [11, §6]. The discussion in [11] highlights the principle of "composition plus hiding" (enforced by Rule (T-cut), cf. Fig. 5) as a feature that distinguishes logically motivated session type systems from other type systems (in particular, the one in [23]), which can type P but also deadlocked variants of it (cf. Example 2.1). However, Caires et al. give no formal comparisons with other classes of deadlock-free processes.

5. Translating $\mathcal{K}$ into $\mathcal{L}$

Having established and characterized the differences between deadlock-free session π-calculus processes in $\mathcal{L}$ and $\mathcal{K}$, in this section we explore how fundamental these differences really are. To this end, we define a type-preserving translation from processes in $\mathcal{K}$ into processes in $\mathcal{L}$.

One leading motivation for looking into a translation is that the separation result established by Corollary 4.1 can be seen as being *subtle*, in the following sense. Consider a process P that belongs to $\mathcal{K}$ but not to $\mu\mathcal{K}$ because one of its subprocesses does not conform to the (restrictive) form of typed parallel composition enforced by Rule (Tπ-PAR+RES) in Fig. 11. The fact that $P \in \mathcal{K} \setminus \mu\mathcal{K}$ implies that P features forms of session cooperation that are intrinsically sequential and admitted by Rules (Tπ-RES) and (Tπ-PAR) in Fig. 6; this means that subprocesses of P must become more independent (concurrent) to be admitted in $\mu\mathcal{K}$. We illustrate this intuition with an example:

Example 5.1 *(Idea of the translation).* Recall process P in Lemma 4.2:

$$P \triangleq (\nu a_1 b_1)(\nu a_2 b_2)(a_1(x).\,\overline{a_2}\langle x\rangle \mid \overline{b_1}\langle \mathbf{n}\rangle.\,b_2(z))$$

We have that $P \in \mathcal{K}$ but $P \notin \mu\mathcal{K}$: each sub-process features two independent sessions occurring in sequence.

Consider now the following variant of P, in which the left subprocess has been kept unchanged, but the right subprocess has been modified to increase concurrency:

$$P' \triangleq (\boldsymbol{\nu} a_2 b_2)((\boldsymbol{\nu} a_1 b_1)(a_1(x).\,\overline{a_2}\langle x\rangle \mid \overline{b_1}\langle \mathbf{n}\rangle.\mathbf{0}) \mid b_2(z).\mathbf{0})$$

Indeed, by replacing $\overline{b_1}\langle \mathbf{n}\rangle.\, b_2(z)$ with $\overline{b_1}\langle \mathbf{n}\rangle.\mathbf{0} \mid b_2(z).\mathbf{0}$, we have that $P' \in \mu\mathcal{K}$.

Here we propose a *translation* that converts any typable session process into a process in $\mathcal{L}$ (i.e. $\mu\mathcal{K}$). The translation, given in § 5.2, follows the idea of Example 5.1: given a parallel process as input, return as output a process in which one of the components is kept unchanged, but the other is translated by using representatives of the sessions implemented in it, composed in parallel. Such parallel representatives are formally defined as *characteristic processes* and *catalyzers*, which we introduce next.

5.1. Characteristic processes and catalyzers

We need some preliminary notions. A *characteristic process* of a session type T represents the smallest process in $\mathcal{L}$ inhabiting it.

Definition 5.1 *(Characteristic processes of a session type).* Given a name x, the set of *characteristic processes* of session type T, denoted $(\!|T|\!)^x$, is inductively defined as follows:

$$\begin{aligned}
(\!|\mathbf{end}|\!)^x &\triangleq \{\mathbf{0}\}\\
(\!|?T'.S|\!)^x &\triangleq \left\{x(y).(P \mid Q) \,:\, P \in (\!|T'|\!)^y \wedge Q \in (\!|S|\!)^x\right\}\\
(\!|!T'.S|\!)^x &\triangleq \left\{\overline{x}(y).(P \mid Q) \,:\, P \in (\!|\overline{T'}|\!)^y \wedge Q \in (\!|S|\!)^x\right\}\\
(\!|\&\{l_i : S_i\}_{i\in I}|\!)^x &\triangleq \left\{x \triangleright \{l_i : P_i\}_{i\in I} \,:\, \forall i \in I.\ P_i \in (\!|S_i|\!)^x\right\}\\
(\!|\oplus\{l_i : S_i\}_{i\in I}|\!)^x &\triangleq \bigcup_{i\in I}\left\{x \triangleleft l_i.P_i \,:\, P_i \in (\!|S_i|\!)^x\right\}
\end{aligned}$$

The previous definition extends to typing contexts by composing in parallel independent characteristic processes, one for each of the session types declared in the context. This reflects that sessions in a context declare independent structures of communication.

Definition 5.2 *(Characteristic processes of a session typing context).* Given a context $\Gamma = w_1{:}T_1, \ldots, w_n{:}T_n$, we shall write $(\!|\Gamma|\!)$ to stand for the set $\left\{(P_1 \mid \cdots \mid P_n) \,:\, P_i \in (\!|T_i|\!)^{w_i}\right\}$.

Characteristic processes are well-typed in the system of § 3.1 (cf. Fig. 5):

Lemma 5.1. *Let T be a session type and Γ be a session context.*

1. *For all $P \in (\!|T|\!)^x$, we have $P \vdash_{\mathrm{LL}} x : [\![T]\!]_\ell$.*
2. *For all $P \in (\!|\Gamma|\!)$, we have $P \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell$.*

Proof. The proof of Part 1 is by induction on the structure of T. The proof of Part 2 is by induction on the size of Γ, using Part 1. See Appendix B.1 (Page 33) for details. □

Let us use '$[\cdot]$' to denote a *hole* and $C[\cdot], C'[\cdot], \ldots$ to denote process contexts (i.e., a process with a hole). Building upon characteristic processes, a *catalyzer* for a typing context is a process context that implements the behaviours it declares.

Definition 5.3 *(Catalyzers of a session typing context).* Given a session typing context Γ, we define its set of associated *catalyzers*, noted $\mathcal{C}_\Gamma$, inductively as follows:

$$\mathcal{C}_\Gamma \triangleq \begin{cases} \{[\cdot]\} & \text{if } \Gamma = \emptyset\\ \left\{(\boldsymbol{\nu} x)(C[\cdot] \mid P) \,:\, C[\cdot] \in \mathcal{C}_{\Gamma'} \wedge P \in (\!|T|\!)^x\right\} & \text{if } \Gamma = \Gamma', x : T \end{cases}$$

Given a context $\Gamma = x_1 : T_1, \ldots, x_n : T_n$, let us write $\overline{\Gamma}$ to denote the context $x_1 : \overline{T_1}, \ldots, x_n : \overline{T_n}$, i.e., the context obtained by "dualising" all the types in Γ. The following statement formalizes the complementarity, in terms of session behaviours, between a well-typed process in $\mathcal{L}$ and its associated catalyzers:

Lemma 5.2 *(Catalyzers preserve typing). Let $P \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell$, $[\![\Gamma']\!]_\ell$ and $C[\cdot] \in \mathcal{C}_{\overline{\Gamma}}$. Then $C[P] \vdash_{\mathrm{LL}} [\![\Gamma']\!]_\ell$.*

Proof. Follows from Definition 5.3, which defines $C[\cdot]$ as a composition of characteristic processes, and Lemma 5.1 (Part 1), which ensures the appropriate type for each of them. □

5.2. Translating $\mathcal{K}$ into $\mathcal{L}$

Our translation, given in Definition 5.4, transforms a session-typed process in $\mathcal{K}$ into a *set* of $\mathcal{L}$ processes. Unsurprisingly, a delicate point in this translation is the treatment of parallel composition, which follows the intuition of the transformation of P into P' motivated in Example 5.1: keep half of a parallel process unchanged, and increase the parallelism in the other half. This way, our translation returns a set of processes because we wish to account for both these two alternatives, for the sake of generality.

Intuitively, given a well-typed session process $P_1 \mid P_2$, our translation produces two sets of processes: the first one collects processes of the form $Q_1 \mid G_2$, where Q_1 composes the translation of P_1 within an appropriate catalyzer and G_2 is a characteristic process that implements all sessions declared in the typing of P_2. Similarly, the second set collects processes of the form $G_1 \mid Q_2$, where Q_2 composes the translation of P_2 within an appropriate catalyzer and G_1 is a characteristic process that implements all sessions declared in the typing for P_1. This way, by translating one subprocess and replacing the other with parallel representatives (G_1 and G_2), the translated processes are more independent, and the circular dependencies—the heart of deadlocked processes—are systematically ruled out.

We require some auxiliary notations.

Notation 5.1 *(Bound names).* Concerning bound names and their session types:

- We annotate bound names with their session types: we write $(\boldsymbol{\nu} x_1 y_1 : T_1)\cdots(\boldsymbol{\nu} x_n y_n : T_n)P$ and $x(y:T).P$, for some session types $T, T_1, \ldots, T_n$.
- We sometimes write $\Gamma, \widetilde{x:T} \vdash_{\mathtt{ST}} P$ as shorthand notation for $\Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash_{\mathtt{ST}} P$. Similarly, we write $(\boldsymbol{\nu}\widetilde{xy} : \widetilde{T})P$ to abbreviate $(\boldsymbol{\nu} x_1 y_1 : T_1)\cdots(\boldsymbol{\nu} x_n y_n : T_n)P$.

Using these notations for bound names, we introduce the following notation for well-typed parallel processes, in which "hidden" sessions are explicitly denoted by brackets:

Notation 5.2 *(Hidden/bracketed sessions).* We shall write

$$\Gamma_1, [\widetilde{x:S}] \star \Gamma_2, [\widetilde{y:T}] \vdash_{\mathtt{ST}} (\boldsymbol{\nu}\widetilde{xy} : \widetilde{S})(P_1 \mid P_2)$$

whenever $\Gamma_1, \Gamma_2 \vdash_{\mathtt{ST}} (\boldsymbol{\nu} x_1 y_1)\cdots(\boldsymbol{\nu} x_n y_n)(P_1 \mid P_2)$ holds with $\Gamma_1, x_1 : S_1, \ldots, x_n : S_n \vdash_{\mathtt{ST}} P_1$, $\Gamma_2, y_1 : T_1, \ldots, y_n : T_n \vdash_{\mathtt{ST}} P_2$, and $S_i = \overline{T_i}$, for all $i \in \{1, \ldots, n\}$.

We are now ready to give the first translation from session processes into $\mathcal{L}$:

Definition 5.4 *(Translation into $\mathcal{L}$).* Let P be such that $\Gamma \vdash_{\mathtt{ST}} P$ and $P \in \mathcal{K}$. The set of $\mathcal{L}$ processes $(\!|\Gamma \vdash_{\mathtt{ST}} P|\!)$ is defined in Fig. 12.

Our translation operates on typing judgements: a well-typed session process is translated using the information declared in its typing context. Although the translation could be defined for arbitrary session processes (even deadlocked ones), membership in $\mathcal{K}$ plays a role in operational correspondence (see below). We discuss the different cases in Fig. 12:

- The process $\mathbf{0}$ is translated into the singleton set $\{\mathbf{0}\}$ provided that the associated typing context Γ contains only completed sessions (recall that Γ^{un} stands for $\mathsf{un}(\Gamma)$).
- The translation of output- and input-prefixed processes is self-explanatory; in the former case, we translate the free output available in $\mathcal{K}$ by exploiting a forwarding process in $\mathcal{L}$ (cf. Definition 4.1). The translation of selection and branching processes also follows expected lines.
- The last case of the definition handles processes in parallel, possibly with restricted sessions; we use Notation 5.2 to make such sessions explicit. As hinted at above, the translation of a parallel process $(\boldsymbol{\nu}\widetilde{xy} : \widetilde{S})(P_1 \mid P_2)$ in $\mathcal{K}$ results into two different sets of $\mathcal{L}$ processes: the first set contains processes of the form $C_1[Q_1] \mid G_2$, where, intuitively:
  - Q_1 belongs to the set that results from translating subprocess P_1 with an appropriate typing judgement, which includes $\widetilde{x:S}$.
  - C_1 belongs to the set of catalyzers that implement the context $\widetilde{x:T}$, i.e., the dual behaviours of the sessions implemented by P_1 (cf. Definition 5.3). This step thus removes the double restriction operator.
  - G_2 belongs to the set of characteristic processes for Γ_2, which describes the sessions implemented by P_2 (cf. Definition 5.2).

  The explanation for the processes of the form $G_1 \mid C_2[Q_2]$ in the second set is completely dual.
  As we will see, processes $C_1[Q_1] \mid G_2$ (and $G_1 \mid C_2[Q_2]$) preserve by construction the typing of $(\boldsymbol{\nu}\widetilde{xy} : \widetilde{S})(P_1 \mid P_2)$: process $C_1[Q_1]$ (resp. $C_2[Q_2]$) is typable with context Γ_1 (resp. Γ_2); process G_2 (resp. G_1) is typable with context Γ_2 (resp. Γ_1)—see Theorem 5.1 below.

$$
\begin{aligned}
(\!|\Gamma^{\mathsf{un}} \vdash_{\mathrm{ST}} \mathbf{0}|\!) &\triangleq \{\mathbf{0}\}\\
(\!|\Gamma, x : !T.S, v : T \vdash_{\mathrm{ST}} \overline{x}\langle v\rangle.P'|\!) &\triangleq \{\overline{x}(z).([v \leftrightarrow z] \mid Q) \,:\, Q \in (\!|\Gamma, x : S \vdash_{\mathrm{ST}} P'|\!)\}\\
(\!|\Gamma_1, \Gamma_2, x : !T.S \vdash_{\mathrm{ST}} (\boldsymbol{\nu} zy)\overline{x}\langle y\rangle.(P_1 \mid P_2)|\!) &\triangleq\\
&\{\overline{x}(y).(Q_1 \mid Q_2) \,:\, Q_1 \in (\!|\Gamma_1, z : \overline{T} \vdash_{\mathrm{ST}} P_1|\!) \wedge Q_2 \in (\!|\Gamma_2, x : S \vdash_{\mathrm{ST}} P_2|\!)\}\\
(\!|\Gamma, x : ?T.S \vdash_{\mathrm{ST}} x(y : T).P'|\!) &\triangleq \{x(y).Q \,:\, Q \in (\!|\Gamma, x : S, y : T \vdash_{\mathrm{ST}} P'|\!)\}\\
(\!|\Gamma, x : \oplus\{l_i : S_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleleft l_j.P'|\!) &\triangleq \{x \triangleleft l_j.Q \,:\, Q \in (\!|\Gamma, x : S_j \vdash_{\mathrm{ST}} P'|\!)\}\\
(\!|\Gamma, x : \&\{l_i : S_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleright \{l_i : P_i\}_{i\in I}|\!) &\triangleq \{x \triangleright \{l_i : Q_i\}_{i\in I} \,:\, Q_i \in (\!|\Gamma, x : S_i \vdash_{\mathrm{ST}} P_i|\!)\}\\
(\!|\Gamma_1, [\widetilde{x : S}] \star \Gamma_2, [\widetilde{y : T}] \vdash_{\mathrm{ST}} (\boldsymbol{\nu}\widetilde{x}\widetilde{y} : \widetilde{S})(P_1 \mid P_2)|\!) &\triangleq\\
&\{C_1[Q_1] \mid G_2 \,:\, Q_1 \in (\!|\Gamma_1, \widetilde{x : S} \vdash_{\mathrm{ST}} P_1|\!),\ C_1 \in \mathcal{C}_{\widetilde{x:T}},\ G_2 \in \langle\!|\Gamma_2|\!\rangle\}\\
&\cup\\
&\{G_1 \mid C_2[Q_2] \,:\, Q_2 \in (\!|\Gamma_2, \widetilde{y : T} \vdash_{\mathrm{ST}} P_2|\!),\ C_2 \in \mathcal{C}_{\widetilde{y:S}},\ G_1 \in \langle\!|\Gamma_1|\!\rangle\}
\end{aligned}
$$

Fig. 12. Translation $(\!|\cdot|\!)$ (cf. Definition 5.4).

We illustrate the translation by means of an example.

Example 5.2 *(The translation $(\!|\cdot|\!)$ at work).* Consider again the process P used in Lemma 4.2. Let $T \triangleq !\mathbf{end}.\mathbf{end}$ and $S \triangleq ?\mathbf{end}.\mathbf{end}$. Clearly, $\overline{S} = T$. We have the following derivation.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a_1 : S, a_2 : T \vdash_{\mathrm{ST}} a_1(x).\,\overline{a_2}\langle x\rangle.\mathbf{0}}\,(\text{T-In}) \qquad \dfrac{}{b_1 : T, b_2 : S, \mathbf{n} : \mathbf{end} \vdash_{\mathrm{ST}} \overline{b_1}\langle \mathbf{n}\rangle.\,b_2(z).\mathbf{0}}\,(\text{T-Out})}{a_2 : T, b_2 : S, a_1 : S, b_1 : T, \mathbf{n} : \mathbf{end} \vdash_{\mathrm{ST}} a_1(x).\,\overline{a_2}\langle x\rangle.\mathbf{0} \mid \overline{b_1}\langle \mathbf{n}\rangle.\,b_2(z).\mathbf{0}}\,(\text{T-Par})}{a_1 : ?\mathbf{end}.\mathbf{end}, b_1 : !\mathbf{end}.\mathbf{end}, \mathbf{n} : \mathbf{end} \vdash_{\mathrm{ST}} (\boldsymbol{\nu} a_2 b_2)(a_1(x).\,\overline{a_2}\langle x\rangle.\mathbf{0} \mid \overline{b_1}\langle \mathbf{n}\rangle.\,b_2(z).\mathbf{0})}\,(\text{T-Res})}{\mathbf{n} : \mathbf{end} \vdash_{\mathrm{ST}} (\boldsymbol{\nu} a_1 b_1)(\boldsymbol{\nu} a_2 b_2)(a_1(x).\,\overline{a_2}\langle x\rangle.\mathbf{0} \mid \overline{b_1}\langle \mathbf{n}\rangle.\,b_2(z).\mathbf{0})}\,(\text{T-Res})
$$

Before detailing the set $(\!|\mathbf{n} : \mathbf{end} \vdash_{\mathrm{ST}} P_2|\!)$, we spell out the main ingredients required:

$$
\begin{aligned}
(\!|a_1 : S, a_2 : T \vdash_{\mathrm{ST}} a_1(x).\,\overline{a_2}\langle x\rangle.\mathbf{0}|\!) &= \{a_1(x).\overline{a_2}(z).([x \leftrightarrow z] \mid \mathbf{0})\}\\
(\!|b_1 : T, b_2 : S, \mathbf{n} : \mathbf{end} \vdash_{\mathrm{ST}} \overline{b_1}\langle \mathbf{n}\rangle.\,b_2(z).\mathbf{0}|\!) &= \{\overline{b_1}(u).([\mathbf{n} \leftrightarrow u] \mid b_2(z).\mathbf{0})\}\\
\langle\!|T|\!\rangle^x &= \{\overline{x}(z).(\mathbf{0} \mid \mathbf{0})\}\\
\langle\!|S|\!\rangle^x &= \{x(z).(\mathbf{0} \mid \mathbf{0})\}\\
\mathcal{C}_{a_1:T,a_2:S} &= \{(\boldsymbol{\nu} a_1)(R_1 \mid (\boldsymbol{\nu} a_2)(R_2 \mid [\cdot])) \,:\, R_1 \in \langle\!|S|\!\rangle^{a_1}, R_2 \in \langle\!|T|\!\rangle^{a_2}\}\\
&= \{(\boldsymbol{\nu} a_1)(a_1(z).(\mathbf{0} \mid \mathbf{0}) \mid (\boldsymbol{\nu} a_2)(\overline{a_2}(z).(\mathbf{0} \mid \mathbf{0}) \mid [\cdot]))\}\\
\mathcal{C}_{b_1:S,b_2:T} &= \{(\boldsymbol{\nu} b_1)(Q_1 \mid (\boldsymbol{\nu} b_2)(Q_2 \mid [\cdot])) \,:\, Q_1 \in \langle\!|T|\!\rangle^{b_1}, Q_2 \in \langle\!|S|\!\rangle^{b_2}\}\\
&= \{(\boldsymbol{\nu} b_1)(\overline{b_1}(z).(\mathbf{0} \mid \mathbf{0}) \mid (\boldsymbol{\nu} b_2)(b_2(z).(\mathbf{0} \mid \mathbf{0}) \mid [\cdot]))\}
\end{aligned}
$$

Exploiting Notation 5.2, judgement $\mathbf{n} : \mathbf{end} \vdash_{\mathrm{ST}} P$ can be written as

$$[a_1 : S, a_2 : T] \star \mathbf{n} : \mathbf{end}, [b_1 : T, b_2 : S] \vdash_{\mathrm{ST}} (\boldsymbol{\nu} a_1 b_1)(\boldsymbol{\nu} a_2 b_2)(a_1(x).\,\overline{a_2}\langle x\rangle.\mathbf{0} \mid \overline{b_1}\langle \mathbf{n}\rangle.\,b_2(z).\mathbf{0})$$

We may now define the translation of P into $\mathcal{L}$:

$$
\begin{aligned}
&(\!|[a_1 : S, a_2 : T] \star \mathbf{n} : \mathbf{end}, [b_1 : T, b_2 : S] \vdash_{\mathrm{ST}} (\boldsymbol{\nu} a_1 b_1)(\boldsymbol{\nu} a_2 b_2)(a_1(x).\,\overline{a_2}\langle x\rangle.\mathbf{0} \mid \overline{b_1}\langle \mathbf{n}\rangle.\,b_2(z).\mathbf{0})|\!)\\
=\ &\{C_1[Q_1] \mid \langle\!|\mathbf{n} : \mathbf{end}|\!\rangle^{\mathbf{n}} \ : C_1 \in \mathcal{C}_{a_1:T,a_2:S},\ Q_1 \in (\!|a_1 : S, a_2 : T \vdash_{\mathrm{ST}} a_1(x).\,\overline{a_2}\langle x\rangle.\mathbf{0}|\!)\}\\
&\cup \{C_2[Q_2] \,:\, C_2 \in \mathcal{C}_{b_1:S,b_2:T},\ Q_2 \in (\!|b_1 : T, b_2 : S, \mathbf{n} : \mathbf{end} \vdash_{\mathrm{ST}} \overline{b_1}\langle \mathbf{n}\rangle.\,b_2(z).\mathbf{0}|\!)\}\\
=\ &\{(\boldsymbol{\nu} a_1)(\overline{a_1}(z).(\mathbf{0} \mid \mathbf{0}) \mid (\boldsymbol{\nu} a_2)(a_2(z).(\mathbf{0} \mid \mathbf{0}) \mid a_1(x).\overline{a_2}(z).([x \leftrightarrow z] \mid \mathbf{0}))) \mid \langle\!|\mathbf{n} : \mathbf{end}|\!\rangle^{\mathbf{n}},\\
&\quad (\boldsymbol{\nu} b_1)(b_1(z).(\mathbf{0} \mid \mathbf{0}) \mid (\boldsymbol{\nu} b_2)(\overline{b_2}(z).(\mathbf{0} \mid \mathbf{0}) \mid \overline{b_1}(u).([\mathbf{n} \leftrightarrow u] \mid b_2(z).\mathbf{0})))\}
\end{aligned}
$$

Above, P has two parallel components and so set $(\!|\mathbf{n} : \mathbf{end} \vdash_{\mathtt{ST}} P|\!)$ has two elements, representing the two different possibilities for "dividing" the sequential structure of P into more parallel processes.

5.2.1. Properties

We present two important results about our translation. First, it is type preserving, up to the encoding of types given in Fig. 10:

Theorem 5.1 *(The translation $(\!|\cdot|\!)$ is type preserving). Let $\Gamma \vdash_{\mathtt{ST}} P$. Then, for all $Q \in (\!|\Gamma \vdash_{\mathtt{ST}} P|\!)$, we have that $Q \vdash_{\mathtt{LL}} [\![\Gamma]\!]_\ell$.*

Proof. By induction on the derivation $\Gamma \vdash_{\mathtt{ST}} P$. See Appendix B.2 (Page 34) for details. □

Theorem 5.1 is meaningful, for it says that the session type "interface" of a process (i.e., the set of sessions it implements) is not modified by the translation. That is, $(\!|\cdot|\!)$ modifies the process structure by closely following the session typing discipline.

To establish properties with respect to reduction, we start with a useful notation:

Notation 5.3. Let us write $\Gamma \vdash_{\mathtt{ST}} P_1, P_2$ whenever both $\Gamma \vdash_{\mathtt{ST}} P_1$ and $\Gamma \vdash_{\mathtt{ST}} P_2$ hold. Similarly, let us write $P_1, P_2 \vdash_{\mathtt{LL}} \Gamma$, whenever both $P_1 \vdash_{\mathtt{LL}} \Gamma$ and $P_2 \vdash_{\mathtt{LL}} \Gamma$ hold.

Before showing how our translation satisfies an operational correspondence result (cf. Theorem 5.2), we illustrate the need for a useful auxiliary definition, which will allow us to relate processes typable under the same typing context but featuring a different parallel structure (cf. Definition 5.5).

Example 5.3 *(The parallel structure of typable processes).* Let $P \vdash_{\mathtt{LL}} \Delta$ where $P \triangleq (\boldsymbol{\nu} x)(\overline{x}\langle v\rangle.P_1 \mid x(z).P_2)$ and $\Delta = \Delta_1, \Delta_2, v : T$. Consider the following reduction from P, obtained using the rules in Fig. 4:

$$(\boldsymbol{\nu} x)(\underbrace{\overline{x}\langle v\rangle.P_1}_{\Delta_1,\, v:T} \mid \underbrace{x(z).P_2}_{\Delta_2}) \to (\boldsymbol{\nu} x)(\underbrace{P_1}_{\Delta_1} \mid \underbrace{P_2[v/z]}_{\Delta_2,\, v:T}) \triangleq Q$$

Here we are using a free output process, namely $\overline{x}\langle v\rangle.P_1$, as a shortcut for $\overline{x}(z).([v \leftrightarrow z] \mid P_1)$; recall that this process is typable using Rules (T-id) and (T-⊗) in Fig. 5. Observe that name z is free with scope P_2; for typing to hold, it must be the case that $z : T$ in P_2.

By Theorem 3.1, $Q \vdash_{\mathtt{LL}} \Delta_1, \Delta_2, v : T$. Let us consider the typing of P and Q in relation to their parallel structure. We can notice that even though the typing context Δ remains the same under reduction, the "parallel decomposition" changes after reduction, as highlighted by the under brackets. In particular, the type assignment $v : T$, at first related to the left-hand side component in P, "jumps" after reduction to the right-hand side component in Q. This phenomenon is due to value passing: after reduction, a substitution occurs in the continuation process P_2, where v might be used in different ways. For instance, v could occur within a sequential position within P_2 or at top-level in one of its parallel subprocesses.

Now, consider process Q', which contains subprocesses P_1 and P_2 from Q but has a different form of parallelism:

$$Q' \triangleq (\boldsymbol{\nu} x)(\underbrace{P_1}_{\Delta_1} \mid \underbrace{(\boldsymbol{\nu} z)(P_2 \mid P_z)}_{\Delta_2} \mid \underbrace{P_v}_{v:T})$$

Here, P_z and P_v are characteristic processes implementing T along z and v, respectively. Clearly, $Q' \vdash_{\mathtt{LL}} \Delta_1, \Delta_2, v : T$, just as Q, but its parallel structure is different: in Q' we have two separate subprocesses in parallel, one typed according to Δ_2, the other according to $v : T$. In $(\boldsymbol{\nu} z)(P_2 \mid P_z)$, process P_z provides one of the counterparts for P_2; this interaction is hidden by restriction $(\boldsymbol{\nu} z)$ and so the resulting typing context is Δ_2. By using P_v, the type interface of Q' is exactly as that of Q; by using it in parallel position, their parallel structure will be different whenever z occurs in P_2 in a sequential position.

We are interested in capturing the *parallelization relation* between processes such as Q and Q' in Example 5.3, because it serves to explain how translated processes behave under reduction. This relation is formalised in Definition 5.5 below:

Definition 5.5 *(Parallelization relation).* Let P and Q be processes such that $P, Q \vdash_{\mathtt{LL}} \Gamma$. We write $P \doteqdot Q$ if and only if there exist processes P_1, P_2, Q_1, Q_2 and contexts Γ_1, Γ_2 such that the following hold:

$$P = P_1 \mid P_2 \qquad Q = Q_1 \mid Q_2 \qquad P_1, Q_1 \vdash_{\mathtt{LL}} \Gamma_1 \qquad P_2, Q_2 \vdash_{\mathtt{LL}} \Gamma_2 \qquad \Gamma = \Gamma_1, \Gamma_2$$

This definition says that two processes P and Q, typed under the typing context Γ, are related by $\doteqdot$, if they can be decomposed into parallel subprocesses, which are typed under the same decomposition of Γ. This way, for processes Q and Q' from Example 5.3, relation $Q \doteqdot Q'$ holds.

By definition, the relation $\div$ is reflexive. It will appear in our operational correspondence result, given next. Below, let $\hookrightarrow$ denote structural congruence (cf § 2.1) extended with a reduction by Rule (R-Fwd) (cf. § 3.1). We may now state:

Theorem 5.2 *(Operational correspondence for $(\!|\cdot|\!)$). Let P be such that $\Gamma \vdash_{\mathtt{ST}} P$ for some typing context Γ. Then, we have:*

1. *If $P \to P'$, then for all $Q \in (\!|\Gamma \vdash_{\mathtt{ST}} P|\!)$ there exist Q', R such that $Q \to\!\!\hookrightarrow Q'$, $Q' \div R$, and $R \in (\!|\Gamma \vdash_{\mathtt{ST}} P'|\!)$.*
2. *If $Q \in (\!|\Gamma \vdash_{\mathtt{ST}} P|\!)$, such that $P \in \mathcal{K}$, and $Q \to\!\!\hookrightarrow Q'$, then there exist P', R such that $P \to P'$, $Q' \div R$, and $R \in (\!|\Gamma \vdash_{\mathtt{ST}} P'|\!)$.*

Proof. By induction on the length of the derivations $P \to P'$ and $Q \to Q'$. See Appendix B.3 for details. □

Part 1 of Theorem 5.2 certifies that our translation tightly preserves the behaviour of the session process given as input. The parallelization relation $\div$ (cf. Definition 5.5) is crucial when the reduction $P \to P'$ involves value passing: in that case, process Q', obtained from the translated process Q, may have a different parallel structure than R—just as Q and Q' from Example 5.3. Part 2 relates the behaviour of a translated process with respect to that of the session process given as input, with $\div$ playing a role similar as in Part 1. Unlike Part 1, in Part 2 we require P to be in $\mathcal{K}$, i.e., P must be deadlock-free for the correspondence to hold. Indeed, if P is not in $\mathcal{K}$ then Q (its deadlock-free, translated variant) could have reductions not enabled in P due to deadlocks.

5.3. Discussion: translating $\mathcal{K}$ into $\mathcal{L}$ exploiting value dependencies

Process prefixes can induce *causality relations* not accounted for by session types. One kind of causality relations are at the heart of deadlocked processes, in which session interleavings in process prefixes leads to circular dependencies between independent sessions (cf. Example 2.1). Another kind of causality relations are the *value dependencies* induced by value exchanges: they occur when a value received in one session is sent along a different one.

Example 5.4 *(A value dependency).* Let P be the process

$$P \triangleq (\boldsymbol{\nu} a_0 b_0)(\overline{a_0}\langle \mathbf{n}\rangle.a_1(u).\overline{a_2}\langle u\rangle.\mathbf{0} \mid b_0(v).(b_2(y).y(x).\mathbf{0} \mid (\boldsymbol{\nu} wz)(\overline{b_1}\langle w\rangle.\overline{z}\langle \mathbf{n}\rangle.\mathbf{0})))$$

Also, let $U \triangleq ?\mathbf{end}.\mathbf{end}$. Consider the typing judgement for the leftmost parallel process $\overline{a_0}\langle \mathbf{n}\rangle.a_1(u).\overline{a_2}\langle u\rangle.\mathbf{0}$:

$$a_0 : !\mathbf{end}.\mathbf{end}, a_1 : ?U.\mathbf{end}, a_2 : !U.\mathbf{end} \vdash_{\mathtt{ST}} \overline{a_0}\langle \mathbf{n}\rangle.a_1(u).\,\overline{a_2}\langle u\rangle.\mathbf{0}$$

Although the typing for a_1 and a_2 states that they are independent sessions, actually they are causally related: the process forwards along a_2 the value of type U received on a_1. By considering the typing for P

$$a_1 : ?U.\mathbf{end}, a_2 : !U.\mathbf{end}, b_2 : ?U.\mathbf{end}, b_1 : !U.\mathbf{end} \vdash_{\mathtt{ST}} P$$

we see that the forwarded value is w, which is delegated by the rightmost parallel process, $(\boldsymbol{\nu} wz)(\overline{b_1}\langle w\rangle.\overline{z}\langle \mathbf{n}\rangle.\mathbf{0})$.

In the terminology of Boreale and Sangiorgi [4], value dependencies are both *subject* and *object* dependencies. In $\overline{a_0}\langle \mathbf{n}\rangle.a_1(u).\overline{a_2}\langle u\rangle.\mathbf{0}$ there is a subject dependency: the input on a_1 enables the output on a_2; there is also an object dependency: the name received on a_1 is used as object in the output on a_2. Indeed, Boreale and Sangiorgi argue that in most cases an object dependency is also a subject dependency.

While intuitive, the translation $(\!|\cdot|\!)$ does not preserve value dependencies: in translating a process with parallel components, a subprocess containing a value dependency can be replaced by an equally typed process in which such a dependency is no longer present:

Example 5.5 *(Value dependencies in $(\!|\cdot|\!)$).* Let process P and type U be as in Example 5.4. Consider the set of processes $(\!|a_1 : ?U.\mathbf{end}, a_2 : !U.\mathbf{end}, b_1 : !U.\mathbf{end}, b_2 : ?U.\mathbf{end} \vdash_{\mathtt{ST}} P|\!)$ obtained using Definition 5.4. One process in this set is the following:

$$Q = G_1 \mid G_2 \mid \mathcal{C}_{b_0:?\mathbf{end}.\mathbf{end}}\big[b_0(v).(b_2(y).y(x).\mathbf{0} \mid (\boldsymbol{\nu} wz)(\overline{b_1}\langle w\rangle.\overline{z}\langle \mathbf{n}\rangle.\mathbf{0}))\big]$$

where $G_1 \in (\!|?U.\mathbf{end}|\!)^{a_1}$ and $G_2 \in (\!|!U.\mathbf{end}|\!)^{a_2}$. Since G_1 and G_2 are independently defined, the name received along a_1 in G_1 cannot be the same session sent along a_2 in G_2. Thus, the value dependence between a_1 and a_2 in P, has disappeared in its translation as Q.

To address this limitation of $(\!|\cdot|\!)$, we have defined an optimized translation that preserves value dependencies. Here we discuss the key elements of this optimization; the full technical development is presented in the online appendix [20]. The optimization is obtained as follows:

1. Detecting value dependencies requires gathering further information on how types are implemented by process prefixes. To this end, we extend the type system of § 2 with *annotated* output and input session types, written $!^n S.T$ and $?^n S.T$ (with $n \geq 0$), which specify the position of an associated prefix within the process. This way, e.g., type $!^0 S.T$ is associated to an output prefix at top-level. Also, we have typing judgements

   $$\Gamma \diamond \Psi \vdash_{\mathrm{ST}} P$$

   where the typing context Γ is as before and Ψ is a *dependency context* that describes all communication prefixes in P and their distance to top-level. This context contains tuples of the form $(a_1, u, 1)$ and $\langle a_2, u, 2\rangle$; the former can be read as "there is an input on a_1 with placeholder u one prefix away from top-level", whereas the latter can be read as "there is an output on a_2 with object u two prefixes away from top-level". Using Ψ, we formally define value dependencies as pairs: we write (a^n, b^m) to denote a value dependency of an (output) prefix along session b on an (input) prefix along session a.
2. We exploit the value dependencies in Ψ to refine the definitions of characteristic processes (Definition 5.1) and catalyzer contexts (Definition 5.3) of a typing context Γ:
   - The set of characteristic processes of Γ that exploits Ψ is denoted $(\!|\Gamma|\!)_\Psi$: it implements value dependencies in Ψ using so-called *bridging sessions* that connect the characteristic processes of the two types involved in the dependency.
   - The set of catalyzer contexts of Γ handles value dependencies in Ψ by "isolating" them using dedicated forwarder processes and characteristic processes.
3. Using these refined definitions, we define the optimized translation $(\!|\cdot|\!)^{\mathrm{V}}$. Main differences with respect to the translation defined in Fig. 12 appear in the translation of parallel processes. As before, process $(\boldsymbol{\nu}\widetilde{xy} : \widetilde{S})(P_1 \mid P_2)$ is translated into processes of two forms: $C_1[Q_1] \mid G_2$ and $G_1 \mid C_2[Q_2]$, where Q_1 (resp. Q_2) stands for the translation of P_1 (resp. P_2). The difference is that the catalyzers C_1, C_2 and the characteristic processes G_1, G_2 are now obtained exploiting value dependencies (as explained in (2)). As $(\!|\cdot|\!)$, the optimized translation $(\!|\cdot|\!)^{\mathrm{V}}$ satisfies type preservation and operational correspondence.

The following example illustrates how $(\!|\cdot|\!)^{\mathrm{V}}$ improves over $(\!|\cdot|\!)$.

Example 5.6 *(Revisiting Examples 5.4 and 5.5).* Let P be as in Example 5.4:

$$P \triangleq (\boldsymbol{\nu} a_0 b_0)(\overline{a_0}\langle \mathbf{n}\rangle.a_1(u).\overline{a_2}\langle u\rangle.\mathbf{0} \mid b_0(v).(b_2(y).y(x).\mathbf{0} \mid (\boldsymbol{\nu} wz)(\overline{b_1}\langle w\rangle.\overline{z}\langle \mathbf{n}\rangle.\mathbf{0})))$$

In the extended type system, we have $\Gamma \diamond \Psi \vdash_{\mathrm{ST}} P$ where

$$\Gamma \triangleq a_1 : ?^1 U.\mathbf{end}, a_2 : !^2 U.\mathbf{end}, b_2 : ?^1 U.\mathbf{end}, b_1 : !^1 U.\mathbf{end}$$

$$\Psi \triangleq \langle a_0, \mathbf{n}, 0\rangle, (a_1, u, 1), \langle a_2, u, 2\rangle, (b_0, v, 0), (b_2, y, 1), (y, x, 2), \langle b_1, w, 1\rangle, \langle z, \mathbf{n}, 2\rangle$$

Using Ψ, we can detect the value dependence (a_1^1, a_2^2). Consider now Q', one particular process included in the set $(\!|\Gamma \diamond \Psi \vdash_{\mathrm{ST}} P|\!)^{\mathrm{V}}$:

$$Q' = (\boldsymbol{\nu} c_{a_1a_2})(G'_{a_1} \mid G'_{a_2}) \mid C_{b_0:?\mathbf{end}.\mathbf{end}}\big[b_0(v).(b_2(y).y(x).\mathbf{0} \mid (\boldsymbol{\nu} wz)(\overline{b_1}\langle w\rangle.\overline{z}\langle \mathbf{n}\rangle.\mathbf{0}))\big]$$

where $G'_{a_1} \in (\!|?U.\mathbf{end}|\!)^{a_1}_{\Psi}$ and $G'_{a_2} \in (\!|!U.\mathbf{end}|\!)^{a_2}_{\Psi}$ are obtained using the refined definition of characteristic processes. With the refined definition, we have that

$$G'_{a_1} = a_1(y).\overline{c_{a_1a_2}}(w).([y \leftrightarrow w] \mid \mathbf{0})$$

$$G'_{a_2} = c_{a_1a_2}(y).\overline{a_2}(w).([y \leftrightarrow w] \mid \mathbf{0})$$

Thus, the characteristic processes of a type are still independently defined but now implement a bridging session $c_{a_1a_2}$, which ensures that the name received along a_1 in G'_{a_1} is the same name outputted along a_2 in G'_{a_2}. The characteristic process of a typing context guarantees that bridging sessions are properly composed. This way, the value dependence between a_1 and a_2, present in P, has been preserved in Q' as a result of the refined translation.

Although intuitive, the way in which $(\!|\cdot|\!)^{\mathrm{V}}$ improves over $(\!|\cdot|\!)$ is somewhat implicit, because value dependencies are accounted for by the revised definitions of characteristic processes and catalyzers as restricted (hidden) sessions. Therefore, the implemented value dependencies in these definitions do not influence the observable behaviour of the translated process. Formalizing the advantages of $(\!|\cdot|\!)^{\mathrm{V}}$ over $(\!|\cdot|\!)$ is an interesting question for future work, because it requires devising new notions of observables for typed processes that "look inside" reductions to ensure that values are forwarded appropriately between the two independent sessions.

6. Discussion

Processes with unbounded behaviour. Our investigation has been motivated by the proliferation of type systems for ensuring safety and liveness properties of mobile, concurrent processes. Different type systems enforce different such properties, which include various forms of (dead)lock-freedom, termination, and confluence. In this work, our criteria have been twofold. On the one hand, we have aimed at obtaining objective formal comparisons between well-established type systems, sticking to their original formulations as much as possible. On the other hand, we have concentrated on the intrinsic challenges of statically enforcing deadlock freedom. We have focused on typed processes without constructs for expressing processes with unbounded behaviour, such as replication or recursion. This focus was useful to be consistent with these criteria; next we discuss some of the issues involved in going beyond this class of finite processes.

In the Curry-Howard correspondences for session types, sharing exponentials ! and ? at the level of propositions/types can be interpreted as input-guarded replication $!x(z).Q$ at the level of proofs/processes. In the classical setting considered here, a channel typed with $!A$ denotes a *server* able to offer an arbitrary number of copies (including zero) of a process with behaviour of type A upon request (an input); dually, a channel typed with $?A$ denotes a *request* to such a server (an output). This requires an additional cut rule for exponential (unrestricted) contexts (denoted Θ) as well as typing rules for realizing the sharing semantics of $!A$ and $?A$. Other rules are adjusted to account for Θ; for instance, the mix rule becomes:

$$\frac{P \vdash_{\mathrm{LL}} \Delta; \Theta \qquad Q \vdash_{\mathrm{LL}} \Delta'; \Theta}{P \mid Q \vdash_{\mathrm{LL}} \Delta, \Delta'; \Theta}$$

The resulting logically justified reduction rule for replication is as follows:

$$(\boldsymbol{\nu} x)(\overline{x}(y).P \mid !x(z).Q) \to (\boldsymbol{\nu} x)((\boldsymbol{\nu} y)(P \mid Q) \mid !x(z).Q) \tag{1}$$

The process before the reduction features one linear cut along session x; after the reduction, the process contains two cuts: the cut on y is linear, whereas the one on x is exponential.

The definition of the typed languages $\mathcal{L}$ and $\mathcal{K}$ (Definition 4.4) can be extended to consider typed processes with replication, which is used in [9,11] (in the input-guarded variant described above) and in [38] (where unguarded replication $!P$ is accounted for). Since the type system $\vdash_{\mathrm{ST}}$ in [57] admits rather expressive forms of recursion (that go well beyond the server-request interactions enabled by input-guarded replication), the class of "full $\mathcal{L}$" processes would be a strict sub-class of session-typed processes.

Now, considering processes with unbounded behaviour entails including *termination* properties into the analysis of (dead)lock-freedom. Crucially, the full Curry-Howard interpretation for session types, including exponentials as explained above, is known to be strongly normalizing and confluent [50,10]. Therefore, unbounded behaviour in full $\mathcal{L}$ concerns unboundedly many copies of finite, deterministic interactive behaviours. The fact that a single type system simultaneously enforces deadlock freedom, termination, and confluence sharply contrasts to the situation for non-logical type systems: to our knowledge, only the hybrid type system in [42] simultaneously ensures these three properties (see below). Clearly, it would be unfair to compare processes that enjoy different properties, i.e., processes in $\mathcal{L}$ against well-typed processes in type systems that enforce some, but not all, of deadlock freedom, termination, and confluence. By focusing on finite processes, we have found a fair ground to objectively compare different type systems.

The integration of non-logical type systems for (dead)lock-freedom, termination, and confluence is far from trivial, and requires advanced mechanisms. Kobayashi and Sangiorgi [42] targeted this goal by defining a parametric hybrid type system based on usages, which enforces the three properties through different methods (not necessarily type systems). As in [36], the syntax of usage types in [42] includes replicated usages $*U$, i.e., unboundedly many parallel copies of usage U. (The authors remark that the recursive usages $\mu\alpha.U$ from [38] are also sound.) To enable fair comparisons against full $\mathcal{L}$, Kobayashi and Sangiorgi's type system should be restricted so that it allows only input-guarded replicated processes (cf. (1)) and simultaneously enforces (dead)lock-freedom, termination, and confluence. Identifying the syntactic/semantic conditions that enable this restriction seems challenging. Defining such a restricted variant of [42] would most likely mean developing a very different type system, therefore departing from the compact abstractions given by usage types. Hence, comparing such a different type system against the canonical language full $\mathcal{L}$ would also be unfair.

We notice that Curry-Howard interpretations of session types have been extended with forms of (co)-recursive types, which extend (full) $\mathcal{L}$ by admitting as typable certain forms of (productive) unbounded behaviour—see the works by Toninho et al. [55] and by Lindley and Morris [43]. Indeed, the framework in [43] can be roughly seen as the extension of the logic-based type system in § 3.1 with co-recursion. Although these extensions bring closer logically motivated and non-logical type systems (which often do not ensure termination), the forms of co-recursive unbounded behaviour enabled by [55,43] preserve the intrinsic confluent, deterministic behaviour inherited from logical foundations. This prevents fair comparisons with type systems for deadlock freedom of unbounded/cyclic communication structures which do not guarantee confluence/determinism, such as those by Giachino et al. [24,33]. In contrast, the type system for deadlock freedom by Padovani [47] ensures a form of *partial confluence*, inherited from [41]. Hence, it would seem that there is common ground for comparing the linear type system in [47] and the logically motivated session type system in [43]. The actual feasibility of relating these different type systems remains unclear, and should be established in future work.

Other Curry-Howard interpretations of session types. As already discussed, our work concerns the Curry-Howard correspondences between (classical) linear logic propositions and session types developed in [9,11,61]. In the following, we briefly discuss our results in the context of other correspondences between (variants of) linear logic and session types [16,34,52].

Dardha and Gay [16] developed Priority-based Classical Processes (PCP), a session type system that rests upon an extension of classical linear logic with Kobayashi's obligations/capabilities, simplified to priorities by following Padovani [47]. Unlike the type system in [61], the type system of PCP admits processes with *safe* cyclic topologies. To this end, PCP cuts ties with the "composition plus hiding" principle that is at the heart of $\mathcal{L}$: the Rule (T-cut) is replaced by rules (T-mix) and (T-cycle) that separately treat parallel composition and restriction, respectively. As a result, a *multicut* rule, which allows composing two processes that may share more than one session, is derivable in PCP. This way, the class of typable processes induced by PCP strictly includes $\mathcal{L}$.

Kokke et al. [34] developed Hypersequent Classical Processes (HCP), an interpretation of session types based on the same classical linear logic as in [61] but with a hypersequent presentation. The design of HCP neatly induces a labelled transition semantics for typed processes; it has been further investigated in, e.g., [52,22]. Also, similarly to PCP, HCP includes separate typing rules for restriction and parallel composition. However, HCP cannot type the kind of safe cyclic process topologies that are typable in PCP: as such, the separate treatment for restriction and parallel composition (enabled by the hypersequent presentation of CLL) does not enhance by itself the topology of processes, and the typable processes in HCP have a tree topology as in $\mathcal{L}$. We conjecture that our separation and unifying results also hold, with minor modifications, for a variant of $\mathcal{L}$ based on HCP. We leave such technical investigation as future work.

Qian et al. [52] recently proposed Client-Server Linear Logic (CSLL) and a corresponding session type system for π-calculus processes, which uses the hypersequent presentation of [34]. By introducing *coexponentials*, CSLL captures more expressive client-server behaviours than those usually admitted by interpretations of CLL as session types. Their approach consists in abandoning the mix principles as adopted by most systems based on CLL (including the one we define in § 3.1). This way, the mix rules in CSLL only concern disjoint concurrency, as captured by hyperenvironments. As already discussed, in our work the Rule (T-mix) enables us to type the independent parallel composition of processes, as needed in several aspects of our development, such as our separation result based on $\mu\mathcal{K}$. Exploring how to adapt our results to a system without mix principles is an interesting item for future work.

Correctness criteria for the translation. The translation $(\!|\cdot|\!)$ defined in § 5 is supported by an operational correspondence result, Theorem 5.2. Operational correspondence is a well-established correctness criterion, which allows us to certify the quality of a language translation. There exist other correctness criteria, which typically serve different purposes [26,48]. A notable criterion is *full abstraction*, i.e., that a translation from a source to a target language preserves the respective equivalences. While full abstraction is not informative enough to assert the quality of a translation (see [25,49] for a discussion), it is quite appropriate to formalise the transfer of reasoning techniques between different calculi. Studying $(\!|\cdot|\!)$ from the perspective of full abstraction would be insightful and requires developing (typed) behavioural equivalences for $\mathcal{K}$ and $\mathcal{L}$. We leave such technical developments as future work.

7. Related work

The analysis of deadlock freedom in concurrency has a rather long and rich history, as discussed in details in [1]. Focusing on type-based approaches to deadlock freedom for communicating processes, early works are by Kobayashi [35] and by Abramsky et al. [1]. The work in [1] develops a semantic approach to deadlock freedom for asynchronous communicating processes, building upon categorical foundations. The work in [35] proposes a type system for the π-calculus that builds upon two key ideas: (i) the introduction of usage of channels as types (*usage types*), and (ii) the classification of channels into reliable and unreliable (where reliable channels are ensured to have deadlock-free interactions). These ideas have proved rather influential: based on them, a number of extensions and enhancements to type systems for deadlock-free, message-passing processes have been introduced. Kobayashi [39] offers a unified presentation of these developments. The work of Kobayashi and Laneve [33] builds upon notions of usage types and reliable channels for the type-based analysis of unbounded process networks.

The introduction of type systems based on usage types coincided in time with the introduction of (binary) session types as a type-based approach to ensure safe structured communications [28,56,29]. In their original formulation, session types for the π-calculus ensure communication safety and session fidelity, therefore ruling out some (but not all!) of the causes of deadlocked behaviours in communicating processes. In session-based concurrency, deadlocks are due to circular dependencies inside a session, but are also found in subtle entanglements between different protocols/sessions. In particular, session type systems [57] cannot guarantee deadlock freedom of interleaved sessions. To our knowledge, Dezani et al. [15] were the first to address progress/deadlock freedom for session-typed processes. Subsequent works on type-based analyses for deadlock freedom in structured communications include [3,12,7,9,46,60,47].

As discussed in detail in this paper, another approach to deadlock freedom is based on linear logic under a Curry-Howard correspondence [9,61], where circular dependencies of communication in processes are eliminated by design, due to Rule (T-CUT). As already mentioned, the type system PCP [16] allows to type safe cyclic process topologies. Following PCP, Kokke and Dardha [32] define Priority GV (PGV) and its implementation in Linear Haskell [31], which is a core concurrent λ-calculus with priorities, whereas the work of Van den Heuvel and Pérez [59] extends PCP with asynchronous communication. Balzer

et al. [5] develop a type system that enforces deadlock freedom for a language with *manifest sharing*, which supports possibly recursive processes in which cycles arise under an acquire-release discipline for shared channels. We note that none of the above works propose a formal comparison between different type systems for deadlock freedom, as we achieve in this paper.

Building upon a translation first suggested by Kobayashi [39], the work of Dardha et al. [17,13,18] offered a formal relationship between usage types and session types. Such a relationship made it possible to use type-based analysis techniques for usage types in the analysis of session-typed process specifications; this insight was formalized by Carbone et at. [8,14]. The effectiveness of the approach in [8] is supported by informal comparisons with respect to different type systems for deadlock freedom (including those in [15] and [7]) using processes typable in one framework but not in another.

A comparison with the workshop version of our work [19] is relevant. In [19], we introduced the notion of *degree of sharing* to classify processes in $\mathcal{K}$. Intuitively, the degree of sharing quantifies the greatest number of channels shared by parallel components: processes with a degree of sharing equal to 0 do not share any channels (concurrency without interaction), processes with a degree of sharing equal to 1 share at most one session, and so on. The degree of sharing determines a strict hierarchy of processes (denoted $\mathcal{K}_0, \mathcal{K}_1, \ldots$ in [19]). This hierarchy, however, is only static—it is not closed under reduction.

A recent work by Van den Heuvel and Pérez [58] also compares session type systems based on Curry-Howard foundations. In [58], the focus is on the classes of typable processes induced by classical and intuitionistic presentations of linear logic; the main result is that the class induced by intuitionistic linear logic is strictly contained in the class induced by classical linear logic. That is, classical linear logic leads to more permissive session type systems. The developments in this paper involved several process languages, each with its own type system—note that the type system that defines $\mathcal{L}$ (Fig. 5) falls under the classical perspective. In contrast, the comparisons in [58] concern different logically-motivated type systems for the same process language.

Loosely related to our work (in particular to the translation in § 5) is previous work on deadlock resolution in the π-calculus by Giunti and Ravara [27] and on unlocking blocked processes by Francalanza et al. [21]. The approach in [27] relies on a typing algorithm that detects a particular class of deadlocks (so-called self-holding deadlocks), but instead of rejecting the code, fixes it by looking into the session types and producing new safe code that obeys the protocols and is deadlock-free. Building upon [27], the work in [21] investigates methods for resolving circular-wait deadlocks across parallel compositions, with a focus on finite CCS processes.

8. Concluding remarks

We have presented a formal comparison of fundamentally distinct type systems for deadlock-free, session typed π-calculus processes. To the best of our knowledge, ours is the first work to establish precise relationships of this kind. Indeed, prior comparisons between type systems for deadlock freedom are informal, given in terms of specific processes typable in one type system but not in some other.

An immediate difficulty in giving a unified account of different typed frameworks for deadlock freedom is the variety of process languages, type structures, and typing rules that define each framework. Our comparisons involve: the framework of session processes put forward by Vasconcelos [57]; the interpretation of linear logic propositions as session types by Caires [6]; the π-calculus with usage types by Kobayashi in [36]. Finding some common ground for comparing these three frameworks is not trivial—several translations/transformations were required in our developments to account for numerous syntactic differences. We made an effort to follow the exact definitions in each framework. Overall, we believe that we managed to concentrate on essential semantic features of two salient classes of deadlock-free session processes, here defined as $\mathcal{L}$ and $\mathcal{K}$ (cf. Definition 4.4).

One main technical contribution is the identification of the precise conditions under which $\mathcal{L}$ and $\mathcal{K}$ coincide. We introduced $\mu\mathcal{K}$, a strict sub-class of $\mathcal{K}$ that coincides with $\mathcal{L}$. The class $\mu\mathcal{K}$ arises as the specialization of Kobayashi's type system [38] that results from internalizing the key aspects of the Curry-Howard interpretation of session types, most notably, the principle of "composition plus hiding", which allows to compose only processes that share exactly one session. This way, $\mu\mathcal{K}$ represents a new characterization of $\mathcal{L}$, given in terms of a very different type system for deadlock freedom adopting usages, capabilities and obligations, put forward by Kobayashi. The identification of $\mu\mathcal{K}$ and its coincidence with $\mathcal{L}$ thus provide an immediate way of separating the classes of processes in $\mathcal{L}$ and $\mathcal{K}$. As another technical contribution, but on the opposite direction, we determined how to unify these two classes by developing an intuitive translation of processes in $\mathcal{K}$ into processes in $\mathcal{L}$, which addresses the syntactic differences between different classes of deadlock-free processes in a type-respecting manner. Although based on simple ideas, our technical developments substantially clarify our understanding of type systems for liveness properties (such as deadlock freedom) in the context of π-calculus processes.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We would like to thank the anonymous reviewers of previous versions of this paper for their useful and constructive suggestions, which led to substantial improvements. We are also grateful to Luís Caires and Simon J. Gay for their valuable comments and suggestions on prior versions of this manuscript. We thank Bas van den Heuvel for his help in designing Fig. 1.

This work was partially supported by the EU COST Action IC1201 (Behavioural Types for Reliable Large-Scale Software Systems). Dardha has been supported by the UK EPSRC project EP/K034413/1 (From Data Types to Session Types: A Basis for Concurrency and Distribution) and by the EU Horizon 2020 MSCA RISE project 778233 "Behavioural Application Program Interfaces" (BehAPI). Pérez has been partially supported by the Dutch Research Council (NWO) under project No. 016.Vidi.189.046 (Unifying Correctness for Communicating Software). Pérez is also affiliated to the NOVA Laboratory for Computer Science and Informatics, Universidade Nova de Lisboa, Portugal.

Appendix A. Omitted proofs for § 4

A.1. Proof of Theorem 4.1 (Page 18)

We divide the proof into the following two lemmas: Lemma A.2 (see below) and Lemma A.3 (Page 31).

Lemma A.1 *(Substitution lemma for CP). If* $P \vdash_{\text{LL}} \Gamma, x : T$ *and* $v \notin \mathsf{fn}(P)$*, then* $P[v/x] \vdash_{\text{LL}} \Gamma, v : T$.

Lemma A.2. *If* $P \in \mathcal{L}$ *then* $P \in \mu\mathcal{K}$.

Proof. By structural induction on P. By Definition 4.4 and Definition 4.5, we have that

$$\mathcal{L} = \left\{P \in \mathbb{P} \,:\, \exists \Gamma.\ (\Gamma \vdash_{\text{ST}} P \ \wedge\ [\![P]\!]_\ell \vdash_{\text{LL}} [\![\Gamma]\!]_\ell)\right\}$$
$$\mu\mathcal{K} = \left\{P \in \mathbb{P} \,:\, \exists \Gamma, f.\ (\Gamma \vdash_{\text{ST}} P \ \wedge\ [\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f} \vdash^{\mu}_{\prec} [\![P]\!]_{\mathsf{u}}^{f})\right\}$$

where $[\![\cdot]\!]_\ell$ is as in Definition 4.1. Following the syntax in Fig. 2, there are seven cases to consider:

1. $P = \mathbf{0}$: By assumption we have both $\Gamma \vdash_{\text{ST}} \mathbf{0}$, for some context Γ (with $\mathsf{un}(\Gamma)$), and $\mathbf{0} \vdash_{\text{LL}} x_1{:}\bullet, \cdots, x_n{:}\bullet$, for some $x_1, \ldots, x_n$ such that $x_i : \mathbf{end} \in \Gamma$, for all $i \in \{1, \ldots, n\}$. Because $\Gamma^{\downarrow} = \emptyset$, we must show that $[\![\emptyset]\!]_{\mathsf{u}}^{f} \vdash^{\mu}_{\prec} [\![\mathbf{0}]\!]_{\mathsf{u}}^{f}$. By Definition 3.11 and Fig. 7, this is the same as showing that $\emptyset \vdash^{\mu}_{\prec} \mathbf{0}$. The thesis then follows immediately from Rule ($\text{T}\pi$-Nil) in Fig. 11.
2. $P = x(y).P'$: By assumption and Definition 3.11, Definition 4.1, and Definition 4.4, we have

$$\Gamma, x : ?T.S \vdash_{\text{ST}} x(y).P' \tag{A.1}$$
$$x(y).[\![P']\!]_\ell \vdash_{\text{LL}} [\![\Gamma]\!]_\ell, x : [\![T]\!]_\ell \otimes [\![S]\!]_\ell$$

for some context Γ and session types S, T. By inversion on typing judgement on (A.1):

$$\frac{\Gamma, x : S, y : T \vdash_{\text{ST}} P'}{\Gamma, x : ?T.S \vdash_{\text{ST}} x(y).P'} \tag{A.2}$$

By Definition 3.11 and Definition 4.5 and by Fig. 7, we then must show:

$$f_x : \mathsf{chan}([\![T]\!]_{\mathsf{u}}, [\![S]\!]_{\mathsf{u}}\,;\, ?_0^0)\ ;_{\prec} [\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f} \vdash^{\mu}_{\prec} f_x(y, c).[\![P']\!]_{\mathsf{u}}^{f,\{x \mapsto c\}}$$

By induction hypothesis on the premise of (A.2) we have:

$$[\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f'}, f'_x : [\![S]\!]_{\mathsf{u}}, f'_y : [\![T]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec} [\![P']\!]_{\mathsf{u}}^{f'} \tag{A.3}$$

for some renaming function f'. We let f be such that $f' = f, \{x \mapsto c\}$. We can then use f to rewrite the above judgement (A.3), as follows:

$$[\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f}, c : [\![S]\!]_{\mathsf{u}}, y : [\![T]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec} [\![P']\!]_{\mathsf{u}}^{f,\{x \mapsto c\}} \tag{A.4}$$

Then, the thesis follows from (A.4) after applying Rule ($\text{T}\pi$-In) in Fig. 11.

3. $P = \overline{x}\langle y\rangle.P'$: By assumption and Definition 3.11, Definition 4.1, and Definition 4.4, we have:

$$\begin{aligned}&\Gamma, x : !T.S, y : T \vdash_{\mathrm{ST}} \overline{x}\langle y\rangle.P' \\ &\overline{x}(z).([z \leftrightarrow y] \mid [\![P']\!]_\ell) \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell, x : [\![\overline{T}]\!]_\ell \otimes [\![S]\!]_\ell, y : [\![T]\!]_\ell\end{aligned} \tag{A.5}$$

for some context Γ and session types S, T. By inversion on typing judgement on (A.5) we have:

$$\frac{\Gamma, x : S \vdash_{\mathrm{ST}} P'}{\Gamma, x : !T.S, y : T \vdash_{\mathrm{ST}} \overline{x}\langle y\rangle.P'} \tag{A.6}$$

By Definition 4.5 and Fig. 7, we must show:

$$f_x : \mathsf{chan}([\![T]\!]_{\mathsf{u}}, [\![\overline{S}]\!]_{\mathsf{u}}\,;\, !_0^0)\,;_\prec \left(y : [\![T]\!]_{\mathsf{u}} \mid [\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f}\right) \vdash^{\mu}_{\prec} (\boldsymbol{\nu} c)\overline{f_x}\langle y, c\rangle.[\![P']\!]_{\mathsf{u}}^{f,\{x\mapsto c\}}$$

By induction hypothesis on the premise of (A.6) we have:

$$[\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f'}, f'_x : [\![S]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec} [\![P']\!]_{\mathsf{u}}^{f'}$$

for some renaming function f'. We let f be such that $f' = f, \{x \mapsto c\}$. We can then rewrite the above judgement in terms of f as follows:

$$[\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f}, c : [\![S]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec} [\![P']\!]_{\mathsf{u}}^{f,\{x\mapsto c\}} \tag{A.7}$$

By applying Rule (Tπ-Var) we can derive both

$$y : [\![T]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec} y : [\![T]\!]_{\mathsf{u}}\ (*) \quad \text{and} \quad c : [\![\overline{S}]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec} c : [\![\overline{S}]\!]_{\mathsf{u}}\ (**)$$

Let $U_1 = u([\![\overline{S}]\!]_{\mathsf{u}})$ and $U_2 = u([\![S]\!]_{\mathsf{u}})$. Lemma 4.4 ensures $\mathsf{rel}(U_1 \mid U_2)$. We can then apply Rule (Tπ-BOut) in Fig. 11 on $(*)$, $(**)$, and the induction hypothesis in (A.7):

$$\frac{y : [\![T]\!]_{\mathsf{u}}, c : [\![\overline{S}]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec} y : [\![T]\!]_{\mathsf{u}}, c : [\![\overline{S}]\!]_{\mathsf{u}} \qquad [\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f}, c : [\![S]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec} [\![P']\!]_{\mathsf{u}}^{f,\{x\mapsto c\}} \qquad \mathsf{rel}(U_1 \mid U_2)}{f_x : \mathsf{chan}([\![T]\!]_{\mathsf{u}}, [\![\overline{S}]\!]_{\mathsf{u}}\,;\, !_0^0)\,;_\prec \left(y : [\![T]\!]_{\mathsf{u}} \mid [\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f}\right) \vdash^{\mu}_{\prec} (\boldsymbol{\nu} c)\overline{f_x}\langle y, c\rangle.[\![P']\!]_{\mathsf{u}}^{f,\{x\mapsto c\}}} \tag{A.8}$$

which concludes this case.

4. $P = x \triangleright \{l_i : P_i\}_{i\in I}$. Then, by assumption and Definition 3.11, Definition 4.1, and Definition 4.4 we have:

$$\begin{aligned}&\Gamma, x : \&\{l_i : T_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleright \{l_i : P_i\}_{i\in I} \\ &\quad x \triangleright \{l_i : P_i\}_{i\in I} \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell, x{:}\&\{l_i : [\![T_i]\!]_\ell\}_{i\in I}\end{aligned} \tag{A.9}$$

for some context Γ and session types T_i for $i \in I$. By inversion on the typing judgement given in (A.9) we have:

$$\frac{\Gamma, x : T_i \vdash_{\mathrm{ST}} P_i \quad \forall i \in I}{\Gamma, x : \&\{l_i : T_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleright \{l_i : P_i\}_{i\in I}} \tag{A.10}$$

By Definition 3.11 and Definition 4.5, and by Fig. 7, we must show:

$$f_x : \mathsf{chan}(\langle l_i : [\![T_i]\!]_{\mathsf{u}}\rangle_{i\in I}\,;\, ?_0^0)\,;_\prec [\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f} \vdash^{\mu}_{\prec} f_x(y).\ \mathbf{case}\ y\ \mathbf{of}\ \{l_i\_c \triangleright [\![P_i]\!]_{\mathsf{u}}^{f,\{x\mapsto c\}}\}_{i\in I}$$

By induction hypothesis on the premise of (A.10) we have:

$$[\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f'}, f'_x : [\![T_i]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec} [\![P_i]\!]_{\mathsf{u}}^{f'} \quad \forall i \in I$$

for some renaming function f'. We let f be such that $f' = f, \{x \mapsto c\}$. We can then rewrite the above judgement as follows:

$$[\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f}, c : [\![T_i]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec} [\![P_i]\!]_{\mathsf{u}}^{f,\{x\mapsto c\}} \quad \forall i \in I$$

By applying Rule (Tπ-Var), we obtain $y : \langle l_i : [\![T_i]\!]_{\mathsf{u}}\rangle_{i\in I} \vdash^{\mu}_{\prec} y : \langle l_i : [\![T_i]\!]_{\mathsf{u}}\rangle_{i\in I}$. Then we apply in order Rules (Tπ-Case) and (Tπ-Inp) (cf. Fig. 11) on the judgement above to conclude:

$$f_x : \mathsf{chan}(\langle l_i : [\![T_i]\!]_{\mathsf{u}}\rangle_{i\in I}\,;\, ?_0^0)\,;_\prec [\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f} \vdash^{\mu}_{\prec} f_x(y).\ \mathbf{case}\ y\ \mathbf{of}\ \{l_i\_c \triangleright [\![P_i]\!]_{\mathsf{u}}^{f,\{x\mapsto c\}}\}_{i\in I}$$

5. $P = x \triangleleft l_j.P_j$. Then, by assumption and Definition 3.11, Definition 4.1, and Definition 4.4, we have:

$$\begin{aligned}&\Gamma, x : \oplus\{l_i : T_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleleft l_j.P_j \\ &\quad x \triangleleft l_j.[\![P_j]\!]_\ell \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell, x{:}\oplus\{l_i : [\![T_i]\!]_\ell\}_{i\in I}\end{aligned} \tag{A.11}$$

for some context Γ and session types T_i, for $i \in I$. By inversion on typing judgement on (A.11) we have:

$$\frac{\Gamma, x : T_j \vdash_{\mathrm{ST}} P_j \quad \exists j \in I}{\Gamma, x : \oplus\{l_i : T_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleleft l_j.P_j} \tag{A.12}$$

By Definition 4.5 and Fig. 7, we must show:

$$f_x : \mathsf{chan}(\langle l_i : [\![\overline{T_i}]\!]_{\mathsf{u}}\rangle_{i\in I}\,;\, !^0_0) \,;_\prec [\![\Gamma^{\downarrow}]\!]^f_{\mathsf{u}} \vdash^{\mu}_{\prec} (\boldsymbol{\nu} c)\overline{f_x}\langle l_j\_c\rangle.[\![P_j]\!]^{f,\{x\mapsto c\}}_{\mathsf{u}}$$

By induction hypothesis on the premise of (A.12) we have:

$$[\![\Gamma^{\downarrow}]\!]^{f'}_{\mathsf{u}}, f'_x : [\![T_j]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec} [\![P_j]\!]^{f'}_{\mathsf{u}}$$

for some renaming function f'. We let f be such that $f' = f, \{x \mapsto c\}$. We can then rewrite the above judgement as follows:

$$[\![\Gamma^{\downarrow}]\!]^{f}_{\mathsf{u}}, c : [\![T_j]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec} [\![P_j]\!]^{f,\{x\mapsto c\}}_{\mathsf{u}} \tag{A.13}$$

By applying Rules (Tπ-Var) and (Tπ-LVal) we obtain:

$$c : [\![\overline{T_j}]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec} l_j\_c : \langle l_i : [\![\overline{T_i}]\!]_{\mathsf{u}}\rangle_{i\in I} \tag{A.14}$$

Let $U_1 = u([\![\overline{T_j}]\!]_{\mathsf{u}})$ and $U_2 = u([\![T_j]\!]_{\mathsf{u}})$ Lemma 4.4 ensures $\mathsf{rel}(U_1 \mid U_2)$. We can then apply Rule (Tπ-BOut) (cf. Fig. 11), without a free channel (only a bound one), on (A.13) and (A.14). We have:

$$\frac{c : [\![\overline{T_j}]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec} l_j\_c : \langle l_i : [\![\overline{T_i}]\!]_{\mathsf{u}}\rangle_{i\in I} \qquad [\![\Gamma^{\downarrow}]\!]^{f}_{\mathsf{u}}, c : [\![T_j]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec} [\![P_j]\!]^{f,\{x\mapsto c\}}_{\mathsf{u}} \qquad \mathsf{rel}(U_1 \mid U_2)}{f_x : \mathsf{chan}(\langle l_i : [\![\overline{T_i}]\!]_{\mathsf{u}}\rangle_{i\in I}\,;\, !^0_0) \,;_\prec [\![\Gamma^{\downarrow}]\!]^{f}_{\mathsf{u}} \vdash^{\mu}_{\prec} (\boldsymbol{\nu} c)\overline{f_x}\langle l_j\_c\rangle.[\![P_j]\!]^{f,\{x\mapsto c\}}_{\mathsf{u}}}$$

which concludes this case.

6. $P = (\boldsymbol{\nu} xy)P'$: Then, by assumption and Definition 3.11, 4.1, and 4.4, we have that there exist P_1, P_2 such that $P' = P_1 \mid P_2$ with

$$\begin{aligned}&\Gamma \vdash_{\mathrm{ST}} (\boldsymbol{\nu} xy)(P_1 \mid P_2) \\ &(\boldsymbol{\nu} w)([\![P_1]\!]_\ell[^w/_x] \mid [\![P_2]\!]_\ell[^w/_y]) \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell\end{aligned} \tag{A.15}$$

for some context Γ. By inversion on the typing judgements given in (A.15) we have the following derivation, for some session type T:

$$\frac{\dfrac{\Gamma_1, x : T \vdash_{\mathrm{ST}} P_1 \quad \Gamma_2, y : \overline{T} \vdash_{\mathrm{ST}} P_2}{\Gamma, x : T, y : \overline{T} \vdash_{\mathrm{ST}} P_1 \mid P_2}}{\Gamma \vdash_{\mathrm{ST}} (\boldsymbol{\nu} xy)(P_1 \mid P_2)} \tag{A.16}$$

where $\Gamma = \Gamma_1, \Gamma_2$. Because $P \in \mathcal{L}$, we have that Γ_1 and Γ_2 are disjoint and that channel endpoints x and y (or the single name w in $\mathcal{L}$) form the only shared channel between processes P_1 and P_2. By Definition 4.5 and Fig. 7, we must show:

$$[\![\Gamma^{\downarrow}]\!]^{f}_{\mathsf{u}} \vdash^{\mu}_{\prec} (\boldsymbol{\nu} w)([\![P_1]\!]^{f,\{x\mapsto w\}}_{\mathsf{u}} \mid [\![P_2]\!]^{f,\{y\mapsto w\}}_{\mathsf{u}})$$

By induction hypothesis on the premises of (A.16), we have:

$$[\![\Gamma_1^{\downarrow}]\!]^{f'}_{\mathsf{u}}, f'_x : [\![T]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec_1} [\![P_1]\!]^{f'}_{\mathsf{u}} \quad \text{and} \quad [\![\Gamma_2^{\downarrow}]\!]^{f'}_{\mathsf{u}}, f'_y : [\![\overline{T}]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec_2} [\![P_2]\!]^{f'}_{\mathsf{u}}$$

for some renaming function f'. We let f be such that $f' = f, \{x, y \mapsto w\}$, hence we have $f'_x = f'_y = w$. We can now rewrite the above judgements in terms of f as follows:

$$[\![\Gamma_1^{\downarrow}]\!]^{f}_{\mathsf{u}}, w : [\![T]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec_1} [\![P_1]\!]^{f,\{x\mapsto w\}}_{\mathsf{u}} \quad \text{and} \quad [\![\Gamma_2^{\downarrow}]\!]^{f}_{\mathsf{u}}, w : [\![\overline{T}]\!]_{\mathsf{u}} \vdash^{\mu}_{\prec_2} [\![P_2]\!]^{f,\{y\mapsto w\}}_{\mathsf{u}}$$

Let $U_1 = u([\![T]\!]_{\mathsf{u}})$ and $U_2 = u([\![\overline{T}]\!]_{\mathsf{u}})$. By Lemma 4.4, $\mathsf{rel}(U_1 \mid U_2)$. Combining these facts, we can apply Rule (Tπ-Par+Res) in Fig. 11:

$$\frac{[\![\Gamma_1^{\downarrow}]\!]_{\mathsf{u}}^{f}, w : [\![T]\!]_{\mathsf{u}} \vdash_{\prec_1}^{\mu} [\![P_1]\!]_{\mathsf{u}}^{f,\{x\mapsto w\}} \qquad [\![\Gamma_2^{\downarrow}]\!]_{\mathsf{u}}^{f}, w : [\![\overline{T}]\!]_{\mathsf{u}} \vdash_{\prec_2}^{\mu} [\![P_2]\!]_{\mathsf{u}}^{f,\{y\mapsto w\}} \qquad \mathsf{rel}(U_1 \mid U_2)}{[\![\Gamma_1^{\downarrow}]\!]_{\mathsf{u}}^{f}, [\![\Gamma_2^{\downarrow}]\!]_{\mathsf{u}}^{f} \vdash_{\prec}^{\mu} (\boldsymbol{\nu} w)([\![P_1]\!]_{\mathsf{u}}^{f,\{x\mapsto w\}} \mid [\![P_2]\!]_{\mathsf{u}}^{f,\{y\mapsto w\}})}$$

with $\prec_i = \prec \cup \{(w, y) \mid y \in \mathsf{fn}(P_i) \setminus \{w\}\}$ $(i \in \{1, 2\})$. This completes the proof for this case.

7. $P = P_1 \mid P_2$: This case is similar to the previous one. By assumption we have

$$\Gamma \vdash_{\mathrm{ST}} P_1 \mid P_2 \tag{A.17}$$
$$[\![P_1]\!]_{\ell} \mid [\![P_2]\!]_{\ell} \vdash_{\mathrm{LL}} [\![\Gamma]\!]_{\ell}$$

and inversion on typing on (A.17) we infer:

$$\frac{\Gamma_1 \vdash_{\mathrm{ST}} P_1 \quad \Gamma_2 \vdash_{\mathrm{ST}} P_2}{\Gamma_1, \Gamma_2 \vdash_{\mathrm{ST}} P_1 \mid P_2} \tag{A.18}$$

where $\Gamma = \Gamma_1, \Gamma_2$. As before, because $P \in \mathcal{L}$, we have that Γ_1 and Γ_2 are disjoint: they share no endpoints. By Definition 4.5 and Fig. 7, we must show:

$$[\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f} \vdash_{\prec}^{\mu} [\![P_1]\!]_{\mathsf{u}}^{f} \mid [\![P_2]\!]_{\mathsf{u}}^{f}$$

which follows by induction hypothesis on the premises of (A.18) and by applying Rule (Tπ-IndPar) (cf. Fig. 11). □

Lemma A.3. *If $P \in \mu\mathcal{K}$ then $P \in \mathcal{L}$.*

Proof. By structural induction on P. Recall that by Definition 4.5 and Definition 4.4, we have that

$$\mu\mathcal{K} \triangleq \left\{ P \in \mathbb{P} \;:\; \exists \Gamma, f.\, (\Gamma \vdash_{\mathrm{ST}} P \wedge [\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f} \vdash_{\prec}^{\mu} [\![P]\!]_{\mathsf{u}}^{f}) \right\}$$
$$\mathcal{L} \triangleq \left\{ P \in \mathbb{P} \;:\; \exists \Gamma.\, (\Gamma \vdash_{\mathrm{ST}} P \wedge [\![P]\!]_{\ell} \vdash_{\mathrm{LL}} [\![\Gamma]\!]_{\ell}) \right\}$$

Following the syntax of processes in Fig. 2, there are seven cases to consider:

1. $P = \mathbf{0}$: By assumption we have both $\Gamma \vdash_{\mathrm{ST}} \mathbf{0}$, for some Γ (with $\mathsf{un}(\Gamma)$), and $[\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f} \vdash_{\prec}^{\mu} [\![\mathbf{0}]\!]_{\mathsf{u}}^{f}$. Notice that $\Gamma^{\downarrow} = \emptyset$. We have to show $\mathbf{0} \vdash_{\mathrm{LL}} x_1{:}\bullet, \cdots, x_n{:}\bullet$, for some $x_1, \ldots, x_n$ such that $x_i : \mathbf{end} \in \Gamma$, for all $i \in \{1, \ldots, n\}$. The thesis follows by using Axiom (T-**1**), followed by $n - 1$ applications of Rule (T-⊥) in Fig. 5.
2. $P = x(y).P'$: Then, by assumption, Fig. 7 and Definition 4.5, we have both

$$\Gamma, x : ?T.S \vdash_{\mathrm{ST}} x(y).P' \tag{A.19}$$
$$[\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f}, f_x : \mathsf{chan}([\![T]\!]_{\mathsf{u}}, [\![S]\!]_{\mathsf{u}}\,;\, ?_0^0) \vdash_{\prec}^{\mu} f_x(y, c).[\![P']\!]_{\mathsf{u}}^{f,\{x\mapsto c\}}$$

for some context Γ, session types S, T, and renaming function f. By inversion on typing judgement on (A.19) we have:

$$\frac{\Gamma, x : S, y : T \vdash_{\mathrm{ST}} P'}{\Gamma, x : ?T.S \vdash_{\mathrm{ST}} x(y).P'} \tag{A.20}$$

By Definition 4.1 and Fig. 10, we must show:

$$x(y).[\![P']\!]_{\ell} \vdash_{\mathrm{LL}} [\![\Gamma]\!]_{\ell}, x : [\![T]\!]_{\ell} \otimes [\![S]\!]_{\ell}$$

By induction hypothesis on the premise of (A.20) we have:

$$[\![P']\!]_{\ell} \vdash_{\mathrm{LL}} [\![\Gamma]\!]_{\ell}, x : [\![S]\!]_{\ell}, y : [\![T]\!]_{\ell} \tag{A.21}$$

and the thesis follows easily from (A.21) using Rule (T-⊗) (cf. Fig. 5).
3. $P = \overline{x}\langle y\rangle.P'$: Then, by assumption, Fig. 7 and Definition 4.5, we have both

$$\Gamma, x : !T.S, y : T \vdash_{\mathrm{ST}} \overline{x}\langle y\rangle.P' \tag{A.22}$$
$$[\![\Gamma^{\downarrow}]\!]_{\mathsf{u}}^{f}, f_x : \mathsf{chan}([\![T]\!]_{\mathsf{u}}, [\![S]\!]_{\mathsf{u}}\,;\, !_{\kappa}^0) \vdash_{\prec}^{\mu} (\boldsymbol{\nu} c) f_x\langle y, c\rangle.[\![P']\!]_{\mathsf{u}}^{f,\{x\mapsto c\}}$$

for some typing context Γ, session types S, T and renaming function f. By inversion on typing judgement on (A.22) we have:

$$\frac{\Gamma, x : S \vdash_{\mathrm{ST}} P'}{\Gamma, x : !T.S, y : T \vdash_{\mathrm{ST}} \overline{x}\langle y\rangle.P'} \tag{A.23}$$

By Definition 4.1 and Fig. 10, we must show:

$$\overline{x}(z).([z \leftrightarrow y] \mid [\![P']\!]_\ell) \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell, x : [\![\overline{T}]\!]_\ell \otimes [\![S]\!]_\ell, y : [\![T]\!]_\ell$$

By induction hypothesis on the premise of (A.23) we have:

$$[\![P']\!]_\ell \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell, x : [\![S]\!]_\ell$$

and then we can conclude using Rule (T-⊗) in Fig. 5:

$$\frac{[\![P']\!]_\ell \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell, x : [\![S]\!]_\ell \qquad [y \leftrightarrow z] \vdash_{\mathrm{LL}} y : [\![T]\!]_\ell, z : [\![\overline{T}]\!]_\ell}{\overline{x}(z).([z \leftrightarrow y] \mid [\![P']\!]_\ell) \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell, x : [\![\overline{T}]\!]_\ell \otimes [\![S]\!]_\ell, y : [\![T]\!]_\ell}$$

where $[y \leftrightarrow z] \vdash_{\mathrm{LL}} y : [\![T]\!]_\ell, z : [\![\overline{T}]\!]_\ell$ follows from Rule (T-id).

4. $P = x \triangleleft l_j.P'$: Then, by assumption, Fig. 7, and Definition 4.5, we have both

$$\Gamma, x : \oplus\{l_i : S_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleleft l_j.P' \tag{A.24}$$
$$[\![\Gamma^{\downarrow}]\!]^f_{\mathsf{u}}, f_x : \mathsf{chan}(\langle l_i : [\![\overline{S_i}]\!]_{\mathsf{u}}\rangle_{i\in I}\,;\, !^0_\kappa) \vdash^{\mu}_{\prec} (\boldsymbol{\nu} c)\overline{f_x}\langle l_j\_c\rangle.[\![P']\!]^{f,\{x\mapsto c\}}_{\mathsf{u}}$$

for some typing context Γ, session types S_i for $i \in I$, and renaming function f. By inversion on typing judgement on (A.24) we have:

$$\frac{\Gamma, x : S_j \vdash_{\mathrm{ST}} P' \quad \exists j \in I}{\Gamma, x : \oplus\{l_i : S_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleleft l_j.P'} \tag{A.25}$$

By Definition 4.1 and Fig. 10, we must show:

$$x \triangleleft l_j.[\![P']\!]_\ell \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell, x{:}\oplus\{l_i : [\![S_i]\!]_\ell\}_{i\in I}$$

By induction hypothesis on the premise of (A.25) we have:

$$[\![P']\!]_\ell \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell, x : [\![S_j]\!]_\ell$$

and then we can conclude by using Rule (T-⊕) in Fig. 5:

$$\frac{[\![P']\!]_\ell \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell, x : [\![S_j]\!]_\ell \qquad j \in I}{x \triangleleft l_j.[\![P']\!]_\ell \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell, x{:}\oplus\{l_i : [\![S_i]\!]_\ell\}_{i\in I}}$$

5. $P = x \triangleright \{l_i : P'_i\}_{i\in I}$: Similar to the previous case.
6. $P = (\boldsymbol{\nu} xy)P'$: Then, by assumption, Fig. 7, and Definition 4.5 we have:

$$\Gamma \vdash_{\mathrm{ST}} (\boldsymbol{\nu} xy)P' \tag{A.26}$$
$$[\![\Gamma^{\downarrow}]\!]^f_{\mathsf{u}} \vdash^{\mu}_{\prec} (\boldsymbol{\nu} w)[\![P']\!]^{f,\{x,y\mapsto w\}}_{\mathsf{u}} \tag{A.27}$$

for some context Γ. By Definition 4.1 and Fig. 10, we must show:

$$(\boldsymbol{\nu} w)([\![P_1]\!]_\ell[^w/_x] \mid [\![P_2]\!]_\ell[^w/_y]) \vdash_{\mathrm{LL}} [\![\Gamma_1]\!]_\ell, [\![\Gamma_2]\!]_\ell$$

for some $\Gamma = \Gamma_1, \Gamma_2$.
By inversion on (A.27), using Rule (Tπ-Par+Res) in Fig. 11, we can infer the following:

- Let $w : \mathsf{chan}(\widetilde{T}\,;\, U)$, then $\mathsf{rel}(U)$.
- Since $\mathsf{rel}(U)$, then it must be the case that $U = U_1 \mid U_2$, which in turn implies that $[\![P']\!]^{f,\{x,y\mapsto w\}}_{\mathsf{u}} = [\![P'_1]\!]^{f,\{x\mapsto w\}}_{\mathsf{u}} \mid [\![P'_2]\!]^{f,\{y\mapsto w\}}_{\mathsf{u}}$, for some P'_1, P'_2.
- Processes $[\![P'_1]\!]^{f,\{x\mapsto w\}}_{\mathsf{u}}$ and $[\![P'_2]\!]^{f,\{y\mapsto w\}}_{\mathsf{u}}$ share *exactly* one channel, namely w.

By inversion on the encoding and renaming function f, and on the typing judgement on (A.26) we have:

$$\frac{\dfrac{\Gamma_1, x : S \vdash_{\mathrm{ST}} P_1 \quad \Gamma_2, y : \overline{S} \vdash_{\mathrm{ST}} P_2}{\Gamma_1, x : S, \Gamma_2, y : \overline{S} \vdash_{\mathrm{ST}} P_1 \mid P_2}}{\Gamma_1, \Gamma_2 \vdash_{\mathrm{ST}} (\boldsymbol{\nu} xy)(P_1 \mid P_2)} \tag{A.28}$$

for some session type S and $\Gamma = \Gamma_1, \Gamma_2$. By applying the induction hypothesis on the premises of (A.28), we have both

$$[\![P_1]\!]_\ell \vdash_{\mathrm{LL}} [\![\Gamma_1]\!]_\ell, x : [\![S]\!]_\ell \quad \text{and} \quad [\![P_2]\!]_\ell \vdash_{\mathrm{LL}} [\![\Gamma_2]\!]_\ell, y : [\![\overline{S}]\!]_\ell$$

By Lemma A.1 we obtain:

$$[\![P_1]\!]_\ell[w/x] \vdash_{\mathrm{LL}} [\![\Gamma_1]\!]_\ell, w : [\![S]\!]_\ell \quad \text{and} \quad [\![P_2]\!]_\ell[w/y] \vdash_{\mathrm{LL}} [\![\Gamma_2]\!]_\ell, w : [\![\overline{S}]\!]_\ell$$

We then conclude by using Lemma 4.3 and Rule (Tcut) (cf. Fig. 5):

$$\frac{[\![P_1]\!]_\ell[w/x] \vdash_{\mathrm{LL}} [\![\Gamma_1]\!]_\ell, w : [\![S]\!]_\ell \quad [\![P_2]\!]_\ell[w/y] \vdash_{\mathrm{LL}} [\![\Gamma_2]\!]_\ell, w : [\![\overline{S}]\!]_\ell}{(\boldsymbol{\nu} w)([\![P_1]\!]_\ell[w/x] \mid [\![P_2]\!]_\ell[w/y]) \vdash_{\mathrm{LL}} [\![\Gamma_1]\!]_\ell, [\![\Gamma_2]\!]_\ell}$$

7. $P = P_1 \mid P_2$: similar to the previous case, noticing that since there is no restriction to bind two endpoints together, membership of P in $\mu\mathcal{K}$ relies on Rule (Tπ-InDPar) in Fig. 11 rather than on Rule (Tπ-Par+Res) (as in the previous case). Then, we use Rule (T-mix) (rather than Rule (Tcut)) to type the composition of P_1 and P_2 (cf. Fig. 5). □

Appendix B. Omitted proofs for § 5

B.1. Proof of Lemma 5.1

We repeat the statement in Page 19:

Lemma 5.1. *Let T be a session type and Γ be a session context.*

1. *For all $P \in (\!|T|\!)^x$, we have $P \vdash_{\mathrm{LL}} x : [\![T]\!]_\ell$.*
2. *For all $P \in (\!|\Gamma|\!)$, we have $P \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell$.*

Proof. We consider both parts separately.

Part 1. The proof proceeds by induction on the structure of T. Thus, there are five cases to consider:

1. Case $T = \mathbf{end}$. Then, by Definition 5.1, we have $(\!|\mathbf{end}|\!)^x = \{\mathbf{0}\}$. We conclude by the fact that $[\![\mathbf{end}]\!]_\ell = \bullet$ (cf. Fig. 10) and by Rule (T-**1**) (cf. Fig. 5).
2. Case $T = ?T.S$. Then, by Definition 5.1, P is of the form $x(y).(P_1 \mid P_2)$, with $P_1 \in (\!|T|\!)^y$ and $P_2 \in (\!|S|\!)^x$. By applying the induction hypothesis twice, on T and S, we obtain:

   $$P_1 \vdash_{\mathrm{LL}} y : [\![T]\!]_\ell \qquad P_2 \vdash_{\mathrm{LL}} x : [\![S]\!]_\ell$$

   Now, by applying Rules (T-mix) and (T-$\parr$) (cf. Fig. 5) we have

   $$\frac{\dfrac{P_1 \vdash_{\mathrm{LL}} y : [\![T]\!]_\ell \qquad P_2 \vdash_{\mathrm{LL}} x : [\![S]\!]_\ell}{P_1 \mid P_2 \vdash_{\mathrm{LL}} y : [\![T]\!]_\ell, x : [\![S]\!]_\ell}}{x(y).(P_1 \mid P_2) \vdash_{\mathrm{LL}} x : [\![T]\!]_\ell \parr [\![S]\!]_\ell}$$

   By encoding of types in Fig. 10, we have $[\![?T.S]\!]_\ell = [\![T]\!]_\ell \parr [\![S]\!]_\ell$, which concludes this case.
3. Case $T = !T.S$. Then, by Definition 5.1, P is of the form $\overline{x}(y).(P_1 \mid P_2)$, with $P_1 \in (\!|\overline{T}|\!)^y$ and $P_2 \in (\!|S|\!)^x$. By applying the induction hypothesis twice, on T and S, we obtain

   $$P_1 \vdash_{\mathrm{LL}} y : [\![\overline{T}]\!]_\ell \qquad P_2 \vdash_{\mathrm{LL}} x : [\![S]\!]_\ell$$

   Now, by applying Rule (T-$\otimes$) (cf. Fig. 5) we have

   $$\frac{P \vdash_{\mathrm{LL}} y : [\![\overline{T}]\!]_\ell \qquad Q \vdash_{\mathrm{LL}} x : [\![S]\!]_\ell}{\overline{x}(y).(P \mid Q) \vdash_{\mathrm{LL}} x : [\![\overline{T}]\!]_\ell \otimes [\![S]\!]_\ell}$$

   By encoding of types in Fig. 10, we have $[\![!T.S]\!]_\ell = [\![\overline{T}]\!]_\ell \otimes [\![S]\!]_\ell$ which concludes this case.
4. Case $T = \&\{l_i : S_i\}_{i\in I}$. Then, by Definition 5.1, P is of the form $x \triangleright \{l_i : P_i\}_{i\in I}$, with $P_i \in (\!|S_i|\!)^x$, for all $i \in I$. By induction hypothesis on those S_i, we obtain $P_i \vdash_{\mathrm{LL}} x : [\![S_i]\!]_\ell$, for all $i \in I$. Then, by Rule (T&) (cf. Fig. 5) we have:

   $$\frac{P_i \vdash_{\mathrm{LL}} x : [\![S_i]\!]_\ell \qquad \forall i \in I}{x \triangleright \{l_i : P_i\}_{i\in I} \vdash_{\mathrm{LL}} x : \&\{l_i : [\![S_i]\!]_\ell\}_{i\in I}}$$

   By encoding of types in Fig. 10, $[\![\&\{l_i : S_i\}_{i\in I}]\!]_\ell = \&\{l_i : [\![S_i]\!]_\ell\}_{i\in I}$, which concludes this case.

5. Case $T = \oplus\{l_i : S_i\}_{i\in I}$. Then, by Definition 5.1, P is of the form $x \triangleleft l_j.P_j$, with $P_j \in (\!|S_j|\!)^x$ and $j \in I$. By induction hypothesis on S_j, we obtain $P_j \vdash_{\mathrm{LL}} x : [\![S_j]\!]_\ell$. Then, by Rule (T⊕) (cf. Fig. 5) we have

$$\frac{P_j \vdash_{\mathrm{LL}} x : [\![S_j]\!]_\ell}{x \triangleleft l_j.P_j \vdash_{\mathrm{LL}} x : \oplus\{l_i : [\![S_i]\!]_\ell\}_{i\in I}}$$

By encoding of types in Fig. 10, we have $[\![\oplus\{l_i : S_i\}_{i\in I}]\!]_\ell = \oplus\{l_i : [\![S_i]\!]_\ell\}_{i\in I}$, which concludes this case.

Part 2. Given $\Gamma = w_1 : T_1, \ldots, w_n : T_n$, the proof is by induction on n, the size of Γ. The base case is when $n = 1$: then, by Part 1, $(\!|\Gamma|\!) = (\!|T_1|\!)^{w_1}$, and the thesis follows immediately. The inductive step ($n > 1$) proceeds by using the inductive hypothesis and Rule (T-mix). Let $\Gamma = \Gamma', w_n : T_n$. Then, by inductive hypothesis, $P_i \in (\!|\Gamma'|\!)$ implies $P_i \vdash_{\mathrm{LL}} [\![\Gamma']\!]_\ell$. Recall that $[\![\Gamma']\!]_\ell = w_1{:}[\![T_1]\!]_\ell, \ldots, w_{n-1}{:}[\![T_{n-1}]\!]_\ell$ by Definition 4.2. Also, by Part 1, $P' \in (\!|T_n|\!)^{w_n}$ implies $P' \vdash_{\mathrm{LL}} w_n : [\![T_n]\!]_\ell$. Since by Definition 5.2, $P_i \mid P' \in (\!|\Gamma|\!)$, the thesis follows by composing P_i and P' using Rule (T-mix):

$$\frac{P_i \vdash_{\mathrm{LL}} w_1 : [\![T_1]\!]_\ell, \cdots, w_{n-1} : [\![T_{n-1}]\!]_\ell \qquad P' \vdash_{\mathrm{LL}} w_n : [\![T_n]\!]_\ell}{P_i \mid P' \vdash_{\mathrm{LL}} w_1 : [\![T_1]\!]_\ell, \cdots, w_n : [\![T_n]\!]_\ell}$$

This concludes the proof. □

B.2. Proof of Theorem 5.1

We repeat the statement in Page 22:

Theorem 5.1. *((|·|) is type preserving) Let $\Gamma \vdash_{\mathrm{ST}} P$. Then, for all $Q \in (\!|\Gamma \vdash_{\mathrm{ST}} P|\!)$, we have that $Q \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell$.*

Proof. The proof proceeds by cases on the judgement $\Gamma \vdash_{\mathrm{ST}} P$ used in the translation given in Definition 5.4, and by inversion on the last typing rule applied (given in Fig. 3). There are seven cases to consider:

1. $\Gamma \vdash_{\mathrm{ST}} \mathbf{0}$. Then, by inversion the last rule applied is (T-Nil), with $\mathsf{un}(\Gamma)$. By Definition 5.4 we have $(\!|\Gamma \vdash_{\mathrm{ST}} \mathbf{0}|\!) = \{\mathbf{0}\}$. By applying Rule (T-**1**) (cf. Fig. 5) we have:

$$\frac{}{\mathbf{0} \vdash_{\mathrm{LL}} x : \bullet}\ (\text{T-}\mathbf{1})$$

The thesis follows immediately by the encoding of types $[\![\cdot]\!]_\ell$ in Fig. 10 and Definition 4.2, which ensure that $[\![x : \mathbf{end}]\!]_\ell = x : [\![\mathbf{end}]\!]_\ell = x : \bullet$.

2. $\Gamma', x : !T.S, v : T \vdash_{\mathrm{ST}} \overline{x}\langle v\rangle.P'$, where $\Gamma = \Gamma', x : !T.S, v : T$ and $P = \overline{x}\langle v\rangle.P'$. By inversion the last rule applied is (T-Out):

$$\frac{\Gamma', x : S \vdash_{\mathrm{ST}} P'}{\Gamma', x : !T.S, v : T \vdash_{\mathrm{ST}} \overline{x}\langle v\rangle.P'}\ (\text{T-Out})$$

By Definition 5.4,

$$(\!|\Gamma', x : !T.S, v : T \vdash_{\mathrm{ST}} \overline{x}\langle v\rangle.P'|\!) = \{\overline{x}(z).\big([v \leftrightarrow z] \mid Q\big) \ :\ Q \in (\!|\Gamma', x : S \vdash_{\mathrm{ST}} P'|\!)\}$$

By Rule (T-id) and by Lemma 4.3 we have:

$$\frac{}{[v \leftrightarrow z] \vdash_{\mathrm{LL}} v : [\![T]\!]_\ell, z : [\![\overline{T}]\!]_\ell}\ (\text{T-id}) \tag{B.1}$$

By induction hypothesis, for all $Q \in (\!|\Gamma', x : S \vdash_{\mathrm{ST}} P'|\!)$ we have that $Q \vdash_{\mathrm{LL}} [\![\Gamma']\!]_\ell, x : [\![S]\!]_\ell$. Let Q' be a process in this set. By applying Rule (T-⊗) on Q' and on (B.1) we have

$$\frac{[v \leftrightarrow z] \vdash_{\mathrm{LL}} v : [\![T]\!]_\ell, z : [\![\overline{T}]\!]_\ell \qquad Q' \vdash_{\mathrm{LL}} [\![\Gamma']\!]_\ell, x : [\![S]\!]_\ell}{\overline{x}(z).\big([v \leftrightarrow z] \mid Q'\big) \vdash_{\mathrm{LL}} [\![\Gamma']\!]_\ell, x : [\![\overline{T}]\!]_\ell \otimes [\![S]\!]_\ell, v : [\![T]\!]_\ell}\ (\text{T-}\otimes)$$

By encoding of types $[\![\cdot]\!]_\ell$ in Fig. 10 we have $[\![!T.S]\!]_\ell = [\![\overline{T}]\!]_\ell \otimes [\![S]\!]_\ell$, and by Definition 4.2, we have $[\![\Gamma', x : !T.S, v : T]\!]_\ell = [\![\Gamma']\!]_\ell, x : [\![\overline{T}]\!]_\ell \otimes [\![S]\!]_\ell, v : [\![T]\!]_\ell$, which concludes this case.

3. $\Gamma_1, \Gamma_2, x : !T.S \vdash_{\mathrm{ST}} (\boldsymbol{\nu} zy)\overline{x}\langle y\rangle.(P_1 \mid P_2)$, where $\Gamma = \Gamma_1, \Gamma_2, x : !T.S$. By inversion, this judgement is derived by a sequence of applications of rules, the last rule applied is (T-Res), and before that (T-Out) and (T-Par) as follows:

$$\dfrac{\dfrac{\dfrac{\Gamma_1, z:\overline{T} \vdash_{\mathrm{ST}} P_1 \qquad \Gamma_2, x:S \vdash_{\mathrm{ST}} P_2}{\Gamma_1, z:\overline{T}, \Gamma_2, x:S \vdash_{\mathrm{ST}} P_1 \mid P_2}\,(\text{T-Par})}{\Gamma_1, \Gamma_2, x:!T.S, y:T, z:\overline{T} \vdash_{\mathrm{ST}} \overline{x}\langle y\rangle.(P_1 \mid P_2)}\,(\text{T-Out})}{\Gamma_1, \Gamma_2, x:!T.S \vdash_{\mathrm{ST}} (\boldsymbol{\nu} zy)\overline{x}\langle y\rangle.(P_1 \mid P_2)}\,(\text{T-Res})$$

By Definition 5.4, we have

$$(\!|\Gamma_1, \Gamma_2, x:!T.S \vdash_{\mathrm{ST}} (\boldsymbol{\nu} zy)\overline{x}\langle y\rangle.(P_1 \mid P_2)|\!) = \\ \left\{\overline{x}(z).(Q_1 \mid Q_2) \,:\, Q_1 \in (\!|\Gamma_1, z:\overline{T} \vdash_{\mathrm{ST}} P_1|\!) \;\wedge\; Q_2 \in (\!|\Gamma_2, x:S \vdash_{\mathrm{ST}} P_2|\!)\right\}$$

By induction hypothesis, for all processes

$$Q_1 \in (\!|\Gamma_1, z:\overline{T} \vdash_{\mathrm{ST}} P_1|\!) \text{ we have } Q_1 \vdash_{\mathrm{LL}} [\![\Gamma_1]\!]_\ell, z:[\![\overline{T}]\!]_\ell$$

and

$$Q_2 \in (\!|\Gamma_2, x:S \vdash_{\mathrm{ST}} P_2|\!) \text{ we have } Q_2 \vdash_{\mathrm{LL}} [\![\Gamma_2]\!]_\ell, x:[\![S]\!]_\ell$$

Let Q_1' and Q_2' be processes in the first and second set, respectively. By applying Rule (T-$\otimes$) on Q_1' and Q_2' we have:

$$\dfrac{Q_1' \vdash_{\mathrm{LL}} [\![\Gamma_1]\!]_\ell, z:[\![\overline{T}]\!]_\ell \qquad Q_2' \vdash_{\mathrm{LL}} [\![\Gamma_2]\!]_\ell, x:[\![S]\!]_\ell}{\overline{x}(z).\left(Q_1' \mid Q_2'\right) \vdash_{\mathrm{LL}} [\![\Gamma_1]\!]_\ell, [\![\Gamma_2]\!]_\ell, x:[\![\overline{T}]\!]_\ell \otimes [\![S]\!]_\ell}\,(\text{T-}\otimes)$$

By the encoding of types $[\![\cdot]\!]_\ell$ in Fig. 10 we have $[\![!T.S]\!]_\ell = [\![\overline{T}]\!]_\ell \otimes [\![S]\!]_\ell$, and by Definition 4.2, we have $[\![\Gamma_1, \Gamma_2, x:!T.S]\!]_\ell = [\![\Gamma_1]\!]_\ell, [\![\Gamma_2]\!]_\ell, x:[\![\overline{T}]\!]_\ell \otimes [\![S]\!]_\ell$, which concludes this case.

4. $\Gamma', x:?T.S \vdash_{\mathrm{ST}} x(y:T).P'$, where $\Gamma = \Gamma', x:?T.S$. By inversion, the last typing rule applied is (T-In):

$$\dfrac{\Gamma', x:S, y:T \vdash_{\mathrm{ST}} P'}{\Gamma', x:?T.S \vdash_{\mathrm{ST}} x(y:T).P'}\,(\text{T-In})$$

By Definition 5.4 we have

$$(\!|\Gamma', x:?T.S \vdash_{\mathrm{ST}} x(y:T).P'|\!) = \{x(y).Q \;:\; Q \in (\!|\Gamma', x:S, y:T \vdash_{\mathrm{ST}} P'|\!)\}$$

By induction hypothesis, $Q \in (\!|\Gamma', x:S, y:T \vdash_{\mathrm{ST}} P'|\!)$ implies $Q \vdash_{\mathrm{LL}} [\![\Gamma']\!]_\ell, x:[\![S]\!]_\ell, y:[\![T]\!]_\ell$. Let Q' be a process in this set. By applying Rule (T-$\parr$) on Q':

$$\dfrac{Q' \vdash_{\mathrm{LL}} [\![\Gamma']\!]_\ell, x:[\![S]\!]_\ell, y:[\![T]\!]_\ell}{x(y).Q' \vdash_{\mathrm{LL}} [\![\Gamma']\!]_\ell, x:[\![T]\!]_\ell \parr [\![S]\!]_\ell}\,(\text{T-}\parr)$$

where by the encoding of types $[\![\cdot]\!]_\ell$ in Fig. 10 we have $[\![?T.S]\!]_\ell = [\![T]\!]_\ell \parr [\![S]\!]_\ell$, and by Definition 4.2, we have $[\![\Gamma', x:?T.S]\!]_\ell = [\![\Gamma']\!]_\ell, x:[\![T]\!]_\ell \parr [\![S]\!]_\ell$, which concludes this case.

5. $\Gamma', x:\oplus\{l_i:S_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleleft l_j.P'$, where $\Gamma = \Gamma', x:\oplus\{l_i:S_i\}_{i\in I}$. By inversion, the last typing rule applied is (T-Sel):

$$\dfrac{\Gamma', x:S_j \vdash_{\mathrm{ST}} P' \qquad j \in I}{\Gamma', x:\oplus\{l_i:S_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleleft l_j.P'}\,(\text{T-Sel})$$

By Definition 5.4 we have

$$(\!|\Gamma', x:\oplus\{l_i:S_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleleft l_j.P'|\!) = \{x \triangleleft l_j.Q \;:\; Q \in (\!|\Gamma', x:S_j \vdash_{\mathrm{ST}} P'|\!)\}$$

By induction hypothesis, for all $Q \in (\!|\Gamma', x:S_j \vdash_{\mathrm{ST}} P'|\!)$ we have $Q \vdash_{\mathrm{LL}} [\![\Gamma']\!]_\ell, x:[\![S_j]\!]_\ell$. Let Q' be a process in this set. By applying Rule (T-$\oplus$) on Q' we have:

$$\dfrac{Q' \vdash_{\mathrm{LL}} [\![\Gamma']\!]_\ell, x:[\![S_j]\!]_\ell \qquad j \in I}{x \triangleleft l_j.Q' \vdash_{\mathrm{LL}} [\![\Gamma']\!]_\ell, x:\oplus\{l_i:[\![S_i]\!]_\ell\}_{i\in I}}\,(\text{T-}\oplus)$$

By the encoding of types $[\![\cdot]\!]_\ell$ in Fig. 10, and Definition 4.2, we have

$$[\![\Gamma', x:\oplus\{l_i:S_i\}_{i\in I}]\!]_\ell = [\![\Gamma']\!]_\ell, x:\oplus\{l_i:[\![S_i]\!]_\ell\}_{i\in I}$$

which concludes this case.

6. $\Gamma', x : \&\{l_i : S_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleright \{l_i : P_i\}_{i\in I}$, where $\Gamma = \Gamma', x : \&\{l_i : S_i\}_{i\in I}$. By inversion, the last typing rule applied is (T-Bra):

$$\frac{\Gamma', x : S_i \vdash_{\mathrm{ST}} P_i \qquad \forall i \in I}{\Gamma', x : \&\{l_i : S_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleright \{l_i : P_i\}_{i\in I}} \text{(T-Bra)}$$

By Definition 5.4 we have that

$$(\!|\Gamma', x : \&\{l_i : S_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleright \{l_i : P_i\}_{i\in I}|\!) = \{x \triangleright \{l_i : Q\}_{i\in I} \;:\; Q \in (\!|\Gamma', x : S_i \vdash_{\mathrm{ST}} P_i|\!)\}$$

By induction hypothesis, $Q_i \in (\!|\Gamma', x : S_i \vdash_{\mathrm{ST}} P_i|\!)$ implies $Q_i \vdash_{\mathrm{LL}} [\![\Gamma']\!]_\ell, x : [\![S_i]\!]_\ell$, for all $i \in I$. Let Q_i' be a process in the corresponding set, for all $i \in I$. By applying Rule (T-&) on each of Q_i' we have:

$$\frac{Q_i' \vdash_{\mathrm{LL}} [\![\Gamma']\!]_\ell, x : [\![S_i]\!]_\ell}{x \triangleright \{l_i : Q_i'\}_{i\in I} \vdash_{\mathrm{LL}} [\![\Gamma']\!]_\ell, x : \&\{l_i : [\![S_i]\!]_\ell\}_{i\in I}} \text{(T-\&)}$$

By the encoding of types $[\![\cdot]\!]_\ell$ in Fig. 10 we have $[\![\&\{l_i : S_i\}_{i\in I}]\!]_\ell = \&\{l_i : [\![S_i]\!]_\ell\}_{i\in I}$, and by Definition 4.2, we have $[\![\Gamma', x : \&\{l_i : S_i\}_{i\in I}]\!]_\ell = [\![\Gamma']\!]_\ell, x : \&\{l_i : [\![S_i]\!]_\ell\}_{i\in I}$, which concludes this case.

7. $\Gamma_1, [\widetilde{x : S}], \Gamma_2, [\widetilde{y : T}] \vdash_{\mathrm{ST}} (\boldsymbol{\nu}\widetilde{x}\widetilde{y} : \widetilde{S})(P_1 \mid P_2)$, where by inversion we have $\Gamma_1, \widetilde{x : S} \vdash_{\mathrm{ST}} P_1$ and $\Gamma_2, \widetilde{y : T} \vdash_{\mathrm{ST}} P_2$, and the last typing rule applied is (T-Res), and before that Rule (T-Par) is used, as follows:

$$\frac{\dfrac{\Gamma_1, \widetilde{x : S} \vdash_{\mathrm{ST}} P_1 \;(1) \qquad \Gamma_2, \widetilde{y : T} \vdash_{\mathrm{ST}} P_2 \;(2)}{\Gamma_1, \Gamma_2, \widetilde{x : S}, \widetilde{y : T} \vdash_{\mathrm{ST}} P_1 \mid P_2}\text{(T-Par)}}{\Gamma_1, [\widetilde{x : S}], \Gamma_2, [\widetilde{y : T}] \vdash_{\mathrm{ST}} (\boldsymbol{\nu}\widetilde{x}\widetilde{y} : \widetilde{S})(P_1 \mid P_2)}\text{(T-Res)}$$

Notice that since restriction is the only means of creating dual session channel endpoints (co-variables) and the only restricted names in P are $\widetilde{xy}$, it then follows that $\Gamma_1 \cap \Gamma_2 = \emptyset$. Hence, by the definition of the ',' operator we have that $\Gamma_1, \Gamma_2 = \Gamma_1, \Gamma_2$.
By Definition 5.4, we have that $(\!|\Gamma_1, [\widetilde{x : S}], \Gamma_2, [\widetilde{y : T}] \vdash_{\mathrm{ST}} (\boldsymbol{\nu}\widetilde{xy} : \widetilde{S})(P_1 \mid P_2)|\!)$ is the following set of processes:

$$\left\{C_1[Q_1] \mid G_2 \;:\; C_1 \in \mathcal{C}_{\widetilde{x:T}},\; Q_1 \in (\!|\Gamma_1, \widetilde{x : S} \vdash_{\mathrm{ST}} P_1|\!),\; G_2 \in (\!|\Gamma_2|\!)\right\} \tag{B.2}$$

$$\cup\; \left\{G_1 \mid C_2[Q_2] \;:\; C_2 \in \mathcal{C}_{\widetilde{y:S}},\; Q_2 \in (\!|\Gamma_2, \widetilde{y : T} \vdash_{\mathrm{ST}} P_2|\!),\; G_1 \in (\!|\Gamma_1|\!)\right\} \tag{B.3}$$

We start by inspecting the set of processes in (B.2). By induction hypothesis on the left-hand side premise of Rule (T-Par), marked (1), we have:

$$\text{for all processes } Q \in (\!|\Gamma_1, \widetilde{x : S} \vdash_{\mathrm{ST}} P_1|\!) \text{ we have that } Q \vdash_{\mathrm{LL}} [\![\Gamma_1]\!]_\ell, \widetilde{x : [\![S]\!]_\ell}$$

Let Q' be an arbitrary process in this set. By Lemma 5.2 we have that $C_1[Q'] \vdash_{\mathrm{LL}} [\![\Gamma_1]\!]_\ell$. By Lemma 5.1(b) since $G_2 \in (\!|\Gamma_2|\!)$, we have that $G_2 \vdash_{\mathrm{LL}} [\![\Gamma_2]\!]_\ell$. Since Γ_1 and Γ_2 are disjoint, by Rule (T-mix) we have the following derivation, which concludes the inspection of (B.2):

$$\frac{C_1[Q'] \vdash_{\mathrm{LL}} [\![\Gamma_1]\!]_\ell \qquad G_2 \vdash_{\mathrm{LL}} [\![\Gamma_2]\!]_\ell}{C_1[Q'] \mid G_2 \vdash_{\mathrm{LL}} [\![\Gamma_1]\!]_\ell, [\![\Gamma_2]\!]_\ell} \text{(T-mix)}$$

We inspect now the set of processes in (B.3). By induction hypothesis on the right-hand side premise of Rule (T-Par), marked (2), we have:

$$\text{for all processes } R \in (\!|\Gamma_2, \widetilde{y : T} \vdash_{\mathrm{ST}} R|\!) \text{ we have that } R \vdash_{\mathrm{LL}} [\![\Gamma_2]\!]_\ell, \widetilde{y : [\![T]\!]_\ell}$$

Let R' be an arbitrary process in this set. By Lemma 5.2, $C_2[R'] \vdash_{\mathrm{LL}} [\![\Gamma_2]\!]_\ell$. By Lemma 5.1(b) since $G_1 \in (\!|\Gamma_1|\!)$, we have $G_1 \vdash_{\mathrm{LL}} [\![\Gamma_1]\!]_\ell$. Since Γ_1 and Γ_2 are disjoint, by Rule (T-mix) we have the following derivation:

$$\frac{C_2[R'] \vdash_{\mathrm{LL}} [\![\Gamma_2]\!]_\ell \qquad G_1 \vdash_{\mathrm{LL}} [\![\Gamma_1]\!]_\ell}{C_2[R'] \mid G_1 \vdash_{\mathrm{LL}} [\![\Gamma_1]\!]_\ell, [\![\Gamma_2]\!]_\ell} \text{(T-mix)}$$

We thus conclude that every process belonging to the set in (B.2) or (B.3) is typed under the typing context $[\![\Gamma_1]\!]_\ell, [\![\Gamma_2]\!]_\ell$, concluding this case (and the proof). □

B.3. Proof of Theorem 5.2 (Page 23)

We repeat Definition 5.5 (cf. Page 22):

Definition B.1. Let P, Q be processes such that $P, Q \vdash_{\text{LL}} \Gamma$. We write $P \doteqdot Q$ if and only if there exist P_1, P_2, Q_1, Q_2 and Γ_1, Γ_2 such that the following hold:

$$P = P_1 \mid P_2 \qquad Q = Q_1 \mid Q_2 \qquad P_1, Q_1 \vdash_{\text{LL}} \Gamma_1 \qquad P_2, Q_2 \vdash_{\text{LL}} \Gamma_2 \qquad \Gamma = \Gamma_1, \Gamma_2$$

Lemma B.1 *(Substitution lemma for sessions [57]). If $\Gamma_1 \vdash_{\text{ST}} v : T$ and $\Gamma_2, x : T \vdash_{\text{ST}} P$ and $\Gamma = \Gamma_1, \Gamma_2$, then $\Gamma \vdash_{\text{ST}} P[v/x]$.*

Lemma B.2 *(Substitution lemma for $(\!|\cdot|\!)$). If $P \in (\!|\Gamma, x : T \vdash_{\text{ST}} Q|\!)$ and $v \notin \mathsf{fn}(P, Q)$, then $P[^{v}/_{x}] \in (\!|\Gamma, v : T \vdash_{\text{ST}} Q[v/x]|\!)$.*

Proof. Immediate from Definition 5.4 and Lemma B.1. □

Proposition B.1 *(Composing characteristic processes). Let Γ and T be a typing context and a type, respectively.*

- *If $P_1 \in \langle\!|\Gamma|\!\rangle$ and $P_2 \in \langle\!|T|\!\rangle^{x}$ then $P_1 \mid P_2 \in \langle\!|\Gamma, x : T|\!\rangle$.*
- *If $Q \in \langle\!|\Gamma, x : T|\!\rangle$ then there are Q_1, Q_2 such that $Q = Q_1 \mid Q_2$ with $Q_1 \in \langle\!|\Gamma|\!\rangle$ and $Q_2 \in \langle\!|T|\!\rangle^{x}$.*

Proof. Immediate from Definition 5.2. □

We repeat the statement of the operational correspondence given in Page 23:

Theorem 5.2. *(Operational correspondence for $(\!|\cdot|\!)$) Let P be such that $\Gamma \vdash_{\text{ST}} P$ for some typing context Γ. Then, we have:*

1. *If $P \to P'$, then for all $Q \in (\!|\Gamma \vdash_{\text{ST}} P|\!)$ there exist Q', R such that $Q \to\!\!\!\!\rightarrow Q'$, $Q' \doteqdot R$, and $R \in (\!|\Gamma \vdash_{\text{ST}} P'|\!)$.*
2. *If $Q \in (\!|\Gamma \vdash_{\text{ST}} P|\!)$, such that $P \in \mathcal{K}$, and $Q \to\!\!\!\!\rightarrow Q'$, then there exist P', R such that $P \to P'$, $Q' \doteqdot R$, and $R \in (\!|\Gamma \vdash_{\text{ST}} P'|\!)$.*

Proof. We consider both parts separately.

Part 1. The proof is by induction on the height of the derivation $P \to P'$ (cf. Fig. 2). There are two main cases to consider, which are reductions inferred using Rules (R-Com) and (R-Case); there are also cases corresponding to Rules (R-Par), (R-Res), and (R-Str), which are straightforward via the induction hypothesis. For convenience, below we annotate bound names with their types: this way, e.g., $(\boldsymbol{\nu} xy : S)$ means that $x : S$ and $y : \overline{S}$.

1. Case (R-Com): Then we have:

$$P \triangleq (\boldsymbol{\nu} xy : S')(\overline{x}\langle v\rangle.P_1 \mid y(t : T).P_2) \quad \to \quad (\boldsymbol{\nu} xy : S'')(P_1 \mid P_2[v/t]) \triangleq P'$$

Since $\Gamma \vdash_{\text{ST}} P$, then by inversion we get $S' = !T.S$ for some session types S, T. Then, $S'' = S$. Again by inversion we have the following derivation:

$$\dfrac{\dfrac{\dfrac{\Gamma_1, x : S \vdash_{\text{ST}} P_1}{\Gamma_1, v : T, x : !T.S \vdash_{\text{ST}} \overline{x}\langle v\rangle.P_1}\,(\text{T-Out}) \qquad \dfrac{\Gamma_2, y : \overline{S}, t : T \vdash_{\text{ST}} P_2}{\Gamma_2, y : ?T.\overline{S} \vdash_{\text{ST}} y(t : T).P_2}\,(\text{T-Inp})}{\Gamma_1, v : T, x : !T.S, \Gamma_2, y : ?T.\overline{S} \vdash_{\text{ST}} \overline{x}\langle v\rangle.P_1 \mid y(t : T).P_2}\,(\text{T-Par})}{(\Gamma_1, v : T), \Gamma_2 \vdash_{\text{ST}} (\boldsymbol{\nu} xy : S')(\overline{x}\langle v\rangle.P_1 \mid y(t : T).P_2)}\,(\text{T-Res})$$

By Definition 5.4, the translation of P is as follows:

$$\begin{aligned}(\!|\Gamma \vdash_{\text{ST}} P|\!) &= (\!|(\Gamma_1, v : T), \Gamma_2 \vdash_{\text{ST}} (\boldsymbol{\nu} xy : !T.S)(\overline{x}\langle v\rangle.P_1 \mid y(t : T).P_2)|\!) \\ &= \underbrace{\left\{C_1[Q_1] \mid G_2 \,:\, C_1 \in \mathcal{C}_{x:?T.\overline{S}},\ Q_1 \in (\!|\Gamma_1, x : !T.S, v : T \vdash_{\text{ST}} \overline{x}\langle v\rangle.P_1|\!),\ G_2 \in \langle\!|\Gamma_2|\!\rangle\right\}}_{A_1} \\ &\quad \cup \\ &\quad \underbrace{\left\{G_1 \mid C_2[Q_2] \,:\, C_2 \in \mathcal{C}_{y:!T.S},\ Q_2 \in (\!|\Gamma_2, y : ?T.\overline{S} \vdash_{\text{ST}} y(t : T).P_2|\!),\ G_1 \in \langle\!|\Gamma_1, v : T|\!\rangle\right\}}_{A_2}\end{aligned}$$

where:

$$A_1 = \big\{(\boldsymbol{\nu} x)\big(\overline{x}(w).\big([v \leftrightarrow w] \mid P_1^*\big) \mid P_x\big) \mid G_2 \;:\; P_1^* \in (\!|\Gamma_1, x : S \vdash_{\mathrm{ST}} P_1|\!),\ P_x \in (\!|?T.\overline{S}|\!)^x,\ G_2 \in (\!|\Gamma_2|\!)\big\} \tag{B.4}$$

$$A_2 = \big\{G_1 \mid (\boldsymbol{\nu} y)\big(y(t).P_2^* \mid P_y\big) \;:\; P_2^* \in (\!|\Gamma_2, y : \overline{S}, t : T \vdash_{\mathrm{ST}} P_2|\!),\ P_y \in (\!|!T.S|\!)^y,\ G_1 \in (\!|\Gamma_1, v : T|\!)\big\} \tag{B.5}$$

Before spelling out the translation of P', we record some considerations. By Theorem 5.1, the translation preserves types: $Q \in (\!|\Gamma \vdash_{\mathrm{ST}} P|\!)$ implies $Q \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell$. Since $P \to P'$, Theorem 2.3 ensures $\Gamma \vdash_{\mathrm{ST}} P'$. Again, by Theorem 5.1, $O \in (\!|\Gamma \vdash_{\mathrm{ST}} P'|\!)$ implies $O \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell$. Also, since $\Gamma_2, y : \overline{S}, t : T \vdash_{\mathrm{ST}} P_2$, then by Lemma B.1 we have $\Gamma_2, y : \overline{S}, v : T \vdash_{\mathrm{ST}} P_2[^v/_t]$ and by well-typedness $v \notin \mathrm{fn}(P_2)$. By Definition 5.4, the translation of P' is as follows:

$$\begin{aligned}(\!|\Gamma \vdash_{\mathrm{ST}} P'|\!) &= (\!|\Gamma_1, (\Gamma_2, v : T) \vdash_{\mathrm{ST}} (\boldsymbol{\nu} xy : S)(P_1 \mid P_2[^v/_t])|\!)\\ &= \underbrace{\big\{C_1'[Q_1] \mid G_2' \;:\; C_1' \in \mathcal{C}_{x:\overline{S}},\ Q_1 \in (\!|\Gamma_1, x : S \vdash_{\mathrm{ST}} P_1|\!),\ G_2' \in (\!|\Gamma_2, v : T|\!)\big\}}_{B_1}\\ &\quad \cup\\ &\quad \underbrace{\big\{G_1' \mid C_2'[Q_2] \;:\; C_2' \in \mathcal{C}_{y:S},\ Q_2 \in (\!|\Gamma_2, y : \overline{S}, v : T \vdash_{\mathrm{ST}} P_2[^v/_t]|\!),\ G_1' \in (\!|\Gamma_1|\!)\big\}}_{B_2}\end{aligned}$$

where:

$$B_1 = \big\{G_2' \mid (\boldsymbol{\nu} x)\big(Q_1 \mid P_x'\big) \;:\; P_x' \in (\!|\overline{S}|\!)^x,\ Q_1 \in (\!|\Gamma_1, x : S \vdash_{\mathrm{ST}} P_1|\!),\ G_2' \in (\!|\Gamma_2, v : T|\!)\big\} \tag{B.6}$$

$$B_2 = \big\{G_1' \mid (\boldsymbol{\nu} y)\big(Q_2 \mid P_y'\big) \;:\; P_y' \in (\!|S|\!)^y,\ Q_2 \in (\!|\Gamma_2, y : \overline{S}, v : T \vdash_{\mathrm{ST}} P_2[^v/_t]|\!),\ G_1' \in (\!|\Gamma_1|\!)\big\} \tag{B.7}$$

We now show that every process in $(\!|\Gamma \vdash_{\mathrm{ST}} P|\!)$ reduces into a process in $(\!|\Gamma \vdash_{\mathrm{ST}} P'|\!)$. We address two distinct sub-cases:
(i) We show that every $Q \in A_1$ (cf. Equation (B.4)) reduces to a $Q' \in B_1$ (cf. Equation (B.6));
(ii) We show that every $Q \in A_2$ (cf. Equation (B.5)) reduces to a Q' such that $Q'' \doteq R$, with $R \in B_2$ (cf. Equation (B.7)).

Sub-case (i). Let $Q = G_2 \mid (\boldsymbol{\nu} x)\big(\overline{x}(w).\big([v \leftrightarrow w] \mid P_1^*\big) \mid P_x\big)$ be an arbitrary process in A_1, with $P_x \in (\!|?T.\overline{S}|\!)^x$. By Definition 5.1:

$$(\!|?T.\overline{S}|\!)^x = \big\{x(t).(Q_t \mid Q_x) \;:\; Q_t \in (\!|T|\!)^t \ \wedge\ Q_x \in (\!|\overline{S}|\!)^x\big\}$$

We may then let $P_x = x(t).(Q_t \mid Q_x)$ where $Q_t \in (\!|T|\!)^t$ and $Q_x \in (\!|\overline{S}|\!)^x$. Considering this, and by applying Rules (R-ChCom) and (R-Fwd) (cf. Fig. 4) we have:

$$\begin{aligned} Q &= G_2 \mid (\boldsymbol{\nu} x)\big(\overline{x}(w).\big([v \leftrightarrow w] \mid P_1^*\big) \mid P_x\big)\\ &= G_2 \mid (\boldsymbol{\nu} x)\big(\overline{x}(w).\big([v \leftrightarrow w] \mid P_1^*\big) \mid x(t).(Q_t \mid Q_x)\big)\\ &\to G_2 \mid (\boldsymbol{\nu} x)\big((\boldsymbol{\nu} w)\big([v \leftrightarrow w] \mid P_1^* \mid Q_w \mid Q_x\big)\big)\\ &\hookrightarrow G_2 \mid Q_v \mid (\boldsymbol{\nu} x)\big(P_1^* \mid Q_x\big) \triangleq Q' \end{aligned}$$

(Recall that $\hookrightarrow$ is structural congruence extended with a reduction by Rule (R-Fwd).) We shall show that $Q' \in B_1$. Let us consider/recall the provenance of its different components:
(a) $G_2 \in (\!|\Gamma_2|\!)$
(b) Since Q_v stands for $Q_t[w/t][v/w]$, we have $Q_v \in (\!|T|\!)^v$.
(c) $P_1^* \in (\!|\Gamma_1, x : S \vdash_{\mathrm{ST}} P_1|\!)$.
(d) $Q_x \in (\!|\overline{S}|\!)^x$
By Proposition B.1, Items (a) and (b) above entail:
(e) $G_2 \mid Q_v \in (\!|\Gamma_2, v : T|\!)$
In turn, by considering Items (c), (d), and (e), together with Equation (B.6), it is immediate to see that $Q' \in B_1$. Therefore, $Q' \in (\!|\Gamma \vdash_{\mathrm{ST}} P'|\!)$, as desired.

Sub-case (ii). Let $Q = G_1 \mid (\boldsymbol{\nu}\, y)\big(y(t).P_2^* \mid P_y\big)$ be an arbitrary process in A_2. Since $P_y \in (\!|!T.S|\!)^y$, by Definition 5.1 we have $P_y = \overline{y}(k).(Q_k \mid Q_y)$, where $Q_k \in (\!|\overline{T}|\!)^k$ and $Q_y \in (\!|S|\!)^y$. Considering this, and by applying Rule (R-ChCom) (cf. Fig. 4), we have:

$$\begin{aligned} Q &= G_1 \mid (\boldsymbol{\nu}\, y)\big(y(t).P_2^* \mid P_y\big) \\ &= G_1 \mid (\boldsymbol{\nu}\, y)\big(y(t).P_2^* \mid \overline{y}(k).(Q_k \mid Q_y)\big) \\ &\to G_1 \mid (\boldsymbol{\nu}\, y)\big((\boldsymbol{\nu} k)(P_2^*[k/t] \mid Q_k) \mid Q_y\big) \triangleq Q'' \end{aligned}$$

We shall show that $Q'' \doteq R$, for some $R \in B_2$. Let us consider/recall the provenance of its different components:
(a) $G_1 \in (\!|\Gamma_1, v : T|\!)$
(b) By Lemma B.2, $P_2^*[k/t] \in (\!|\Gamma_2, y : \overline{S}, k : T \vdash_{\mathrm{ST}} P_2[{}^{k}/_{t}]|\!)$.
(c) $Q_k \in (\!|\overline{T}|\!)^k$
(d) $Q_y \in (\!|S|\!)^y$
Furthermore, we can infer:
(e) From (a) and Proposition B.1, there must exist G_1^* and G_v such that $G_1 = G_1^* \mid G_v$, $G_1^* \in (\!|\Gamma_1|\!)$, and $G_v \in (\!|T|\!)^v$.
(f) By combining (b), (c) and (d), together with Proposition B.1, we have that $(\boldsymbol{\nu}\, y)\big((\boldsymbol{\nu} k)(P_2^*[k/t] \mid Q_k) \mid Q_y\big) \in (\!|\Gamma_2|\!)$.
Given this, we can rewrite Q'' as follows:

$$Q'' = G_1^* \mid G_v \mid (\boldsymbol{\nu}\, y)\big((\boldsymbol{\nu} k)(P_2^*[k/t] \mid Q_k) \mid Q_y\big) \tag{B.8}$$

We now consider an arbitrary $R \in B_2$. By Equation (B.7), we have that

$$R \triangleq G_1' \mid (\boldsymbol{\nu}\, y)\big(Q_2 \mid P_y'\big) \tag{B.9}$$

with
(a′) $G_1' \in (\!|\Gamma_1|\!)$
(b′) $Q_2 \in (\!|\Gamma_2, y : \overline{S}, v : T \vdash_{\mathrm{ST}} P_2[{}^{v}/_{t}]|\!)$
(c′) $P_y' \in (\!|S|\!)^y$
By Lemma 5.1 and (c′), we can infer: $P_y' \vdash_{\mathrm{LL}} y : [\![S]\!]_\ell$; and by Theorem 5.1 and (b′), we can infer: $Q_2 \vdash_{\mathrm{LL}} [\![\Gamma_2, y : \overline{S}, v : T]\!]_\ell$. Finally, we have:
(d′) $(\boldsymbol{\nu}\, y)\big(Q_2 \mid P_y'\big) \vdash_{\mathrm{LL}} [\![\Gamma_2]\!]_\ell, [\![v : T]\!]_\ell$
We now compare Q'' and R (as in Equation (B.8) and Equation (B.9), respectively). By Lemma 5.1 and (e) and (f) above, it is easy to see that $Q'', R \vdash_{\mathrm{LL}} [\![\Gamma_1]\!]_\ell, [\![\Gamma_2]\!]_\ell, [\![v : T]\!]_\ell$. Then, by Definition 5.5, we have that $Q'' \doteq R$. Therefore, there is an R such that $Q'' \doteq R$, with $R \in (\!|\Gamma \vdash_{\mathrm{ST}} P'|\!)$, as desired. This concludes the analysis for Case (R-Com).

2. Case (R-Case):

$$P \triangleq (\boldsymbol{\nu} xy : S')(x \triangleleft l_j.Q \mid y \triangleright \{l_i : R_i\}_{i\in I}) \quad \to \quad (\boldsymbol{\nu} xy : S'')(Q \mid R_j) \triangleq P'$$

Since $\Gamma \vdash_{\mathrm{ST}} P$, then by inversion $S' = \oplus\{l_i : S_i\}_{i\in I}$ for some S_i, with $i \in I$. For simplicity, let us write T_i to denote the dual of any S_i. As a result of the reduction, we have $S'' = S_j$ for some $j \in I$. Again by inversion we have the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma_1, x : S_j \vdash_{\mathrm{ST}} Q \quad \exists j \in I}{\Gamma_1, x : \oplus\{l_i : S_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleleft l_j.Q}\,(\text{T-Sel}) \qquad \dfrac{\Gamma_2, y : T_i \vdash_{\mathrm{ST}} R_i \quad \forall i \in I}{\Gamma_2, y : \&\{l_i : T_i\}_{i\in I} \vdash_{\mathrm{ST}} y \triangleright \{l_i : R_i\}_{i\in I}}\,(\text{T-Brch})}{(\Gamma_1, x : \oplus\{l_i : S_i\}_{i\in I}), (\Gamma_2, y : \&\{l_i : T_i\}_{i\in I}) \vdash_{\mathrm{ST}} x \triangleleft l_j.Q \mid y \triangleright \{l_i : R_i\}_{i\in I}}\,(\text{T-Par})}{\Gamma_1, \Gamma_2 \vdash_{\mathrm{ST}} (\boldsymbol{\nu} xy : S')(x \triangleleft l_j.Q \mid y \triangleright \{l_i : R_i\}_{i\in I})}\,(\text{T-Res})$$

By Definition 5.4 the translation of P is as follows:

$$\begin{aligned} (\!|\Gamma \vdash_{\mathrm{ST}} P|\!) &= (\!|\Gamma \vdash_{\mathrm{ST}} (\boldsymbol{\nu} xy : S')(x \triangleleft l_j.Q \mid y \triangleright \{l_i : R_i\}_{i\in I})|\!) \\ &= \underbrace{\big\{C_1[Q_1] \mid G_2 \,:\, C_1 \in \mathcal{C}_{x:\&\{l_i:T_i\}_{i\in I}},\ Q_1 \in (\!|\Gamma_1, x : \oplus\{l_i : S_i\}_{i\in I} \vdash_{\mathrm{ST}} x \triangleleft l_j.Q|\!),\ G_2 \in (\!|\Gamma_2|\!)\big\}}_{A_1} \\ &\quad \cup \\ &\quad \underbrace{\big\{G_1 \mid C_2[Q_2] \,:\, C_2 \in \mathcal{C}_{y:\oplus\{l_i:S_i\}_{i\in I}},\ Q_2 \in (\!|\Gamma_2, y : \&\{l_i : T_i\}_{i\in I} \vdash_{\mathrm{ST}} y \triangleright \{l_i : R_i\}_{i\in I}|\!),\ G_1 \in (\!|\Gamma_1|\!)\big\}}_{A_2} \end{aligned}$$

where:

$$A_1 = \left\{ G_2 \mid (\boldsymbol{\nu} x)\left(x \triangleleft l_j.Q^* \mid P_x\right) \,:\, Q^* \in (\!|\Gamma_1, x : S_j \vdash_{\mathrm{ST}} Q|\!),\ P_x \in (\!|\&\{l_i : T_i\}_{i\in I}|\!)^x,\ G_2 \in (\!|\Gamma_2|\!) \right\} \tag{B.10}$$

$$A_2 = \left\{ G_1 \mid (\boldsymbol{\nu} y)\left(y \triangleright \{l_i : R_i^*\}_{i\in I} \mid P_y\right) \,:\, R_i^* \in (\!|\Gamma_2, y : T_i \vdash_{\mathrm{ST}} R_i|\!),\ P_y \in (\!|\oplus\{l_i : S_i\}_{i\in I}|\!)^y,\ G_1 \in (\!|\Gamma_1|\!) \right\} \tag{B.11}$$

Before spelling out the translation of P', we record some considerations. By Theorem 5.1, $M \in (\!|\Gamma \vdash_{\mathrm{ST}} P|\!)$ implies $M \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell$. Since $P \to P'$, then by Theorem 2.3 we have $\Gamma \vdash_{\mathrm{ST}} P'$. Again, by Theorem 5.1, $O \in (\!|\Gamma \vdash_{\mathrm{ST}} P'|\!)$ implies $O \vdash_{\mathrm{LL}} [\![\Gamma]\!]_\ell$. By Definition 5.4 the encoding of P' is as follows:

$$\begin{aligned}
(\!|\Gamma \vdash_{\mathrm{ST}} P'|\!) &= (\!|\Gamma \vdash_{\mathrm{ST}} (\boldsymbol{\nu} xy : S_j)(Q \mid R_j)|\!) \\
&= \underbrace{\left\{ G_2' \mid C_1'\left[Q^*\right] \,:\, C_1' \in \mathcal{C}_{x:T_j},\ Q^* \in (\!|\Gamma_1, x : S_j \vdash_{\mathrm{ST}} Q|\!),\ G_2' \in (\!|\Gamma_2|\!) \right\}}_{B_1} \\
&\quad \cup \\
&\quad \underbrace{\{ G_1' \mid C_2'\left[R_j^*\right] \,:\, C_2' \in \mathcal{C}_{y:S_j},\ R_j^* \in (\!|\Gamma_2, y : T_j \vdash_{\mathrm{ST}} R_j|\!),\ G_1' \in (\!|\Gamma_1|\!) \}}_{B_2}
\end{aligned}$$

where:

$$B_1 = \left\{ G_2' \mid (\boldsymbol{\nu} x)(Q^* \mid P_x') \,:\, Q^* \in (\!|\Gamma_1, x : S_j \vdash_{\mathrm{ST}} Q|\!),\ P_x' \in (\!|T_j|\!)^x,\ G_2' \in (\!|\Gamma_2|\!) \right\} \tag{B.12}$$

$$B_2 = \{ G_1' \mid (\boldsymbol{\nu} y)(R_j^* \mid P_y') \,:\, R_j^* \in (\!|\Gamma_2, y : T_j \vdash_{\mathrm{ST}} R_j|\!),\ P_y' \in (\!|S_j|\!)^y,\ G_1' \in (\!|\Gamma_1|\!) \} \tag{B.13}$$

We now show that every process in $(\!|\Gamma \vdash_{\mathrm{ST}} P|\!)$ reduces into a process in $(\!|\Gamma \vdash_{\mathrm{ST}} P'|\!)$. We address two distinct sub-cases:
(i) We show that every $Q \in A_1$ (cf. Equation (B.10)) reduces to a $Q' \in B_1$ (cf. Equation (B.12));
(ii) Similarly, we show that every $Q \in A_2$ (cf. Equation (B.11)) reduces to a $Q'' \in B_2$ (cf. Equation (B.13)).

Sub-case (i). Let $Q = G_2 \mid (\boldsymbol{\nu} x)\left(x \triangleleft l_j.Q^* \mid P_x\right)$ be an arbitrary process in A_1, with $Q^* \in (\!|\Gamma_1, x : S_j \vdash_{\mathrm{ST}} Q|\!)$ and $P_x \in (\!|\&\{l_i : T_i\}_{i\in I}|\!)^x$. By Definition 5.1, we may then let $P_x = x \triangleright \{l_i : P_i\}_{i\in I}$, such that $P_i \in (\!|T_i|\!)^x$, for all $i \in I$. By applying Rule (R-ChCase) (cf. Fig. 4), we have:

$$\begin{aligned}
Q \ &\triangleq\ G_2 \mid (\boldsymbol{\nu} x)\left(x \triangleleft l_j.Q^* \mid P_x\right) \\
&\to\ G_2 \mid (\boldsymbol{\nu} x)\left(Q^* \mid P_j\right) \triangleq Q'
\end{aligned}$$

We shall show that $Q' \in B_1$. Let us consider/recall the provenance of its different components:
(a) $G_2 \in (\!|\Gamma_2|\!)$
(b) $Q^* \in (\!|\Gamma_1, x : S_j \vdash_{\mathrm{ST}} Q|\!)$.
(c) $P_j \in (\!|T_j|\!)^x$.
By considering Items (a) – (c), together with Equation (B.12), it is immediate to see that $Q' \in B_1$. Therefore, $Q' \in (\!|\Gamma \vdash_{\mathrm{ST}} P'|\!)$, as desired.

Sub-case (ii). Let $Q = G_1 \mid (\boldsymbol{\nu} y)\left(y \triangleright \{l_i : R_i^*\}_{i\in I} \mid P_y\right)$ be an arbitrary process in A_2, with $R_i^* \in (\!|\Gamma_2, y : T_i \vdash_{\mathrm{ST}} R_i|\!)$ and $P_y \in (\!|\oplus\{l_i : S_i\}_{i\in I}|\!)^y$. By Definition 5.1, P_y is one of the processes in the union $\bigcup_{i\in I}\left\{y \triangleleft l_i.P_i \,:\, P_i \in (\!|S_i|\!)^y\right\}$. We then choose $P_y = y \triangleleft l_j.P_j$ such that $j \in I$ and $P_j \in (\!|S_j|\!)^y$. By applying Rule (R-ChCase) (cf. Fig. 4), we have:

$$\begin{aligned}
Q \ &\triangleq\ G_1 \mid (\boldsymbol{\nu} y)\left(y \triangleright \{l_i : R_i^*\}_{i\in I} \mid y \triangleleft l_j.P_j\right) \\
&\to\ G_1 \mid (\boldsymbol{\nu} y)\left(R_j^* \mid P_j\right) \triangleq Q''
\end{aligned}$$

We shall show that $Q'' \in B_2$. Let us consider/recall the provenance of its different components:
(a) $G_1 \in (\!|\Gamma_1|\!)$
(b) $R_j^* \in (\!|\Gamma_2, y : T_j \vdash_{\mathrm{ST}} R_j|\!)$.
(c) $P_j \in (\!|S_j|\!)^y$.
By considering Items (a) – (c), together with Equation (B.13), it is immediate to see that $Q'' \in B_2$. Therefore, $Q'' \in (\!|\Gamma \vdash_{\mathrm{ST}} P'|\!)$, as desired. This concludes the analysis for Case (R-Case) (and for Part 1).

Part 2. Let $P \in \mathcal{K}$ and $Q \in (\!|\Gamma \vdash_{\mathrm{ST}} P|\!)$. Suppose that $Q \to\!\!\!\!\rightarrow Q'$; we now show that (i) there is a P' such that $P \to P'$ and (ii) $Q' \doteqdot R$, for some $R \in (\!|\Gamma \vdash_{\mathrm{ST}} P'|\!)$.

We first argue for (i), i.e., the existence of a reduction $P \to P'$. Notice that for the reduction(s) $Q \to\!\!\!\!\rightarrow Q'$ to occur, there must exist two complementary prefixes occurring at top level in Q. By Definition 5.4 and, crucially, by the assumption $P \in \mathcal{K}$, the same two prefixes occur at top-level also in P and can reduce; indeed, without the assumption $P \in \mathcal{K}$, it could be that P cannot reduce due to a deadlock. Hence, P can mimic the reduction from Q, up to structural congruence: $P \to P'$.

It then suffices to prove the theorem for $M \equiv P$, in which the two prefixes involved occur in contiguous positions and can reduce.

To address (ii), we now relate P' and Q' by considering two main cases for the reduction originating from Q (cf. Fig. 4): (1) it corresponds to an input-output communication via Rule (R-ChCom); and (2) it corresponds to a selection-branching interaction via Rule (R-ChCase). (The third case, corresponding to Rule (R-ChRes), is straightforward using the induction hypothesis.) We detail these two cases; the analysis largely mirrors the one given in Part 1:

1. Case $M = \mathcal{E}\big[(\boldsymbol{\nu} xy : S')(\overline{x}\langle v\rangle.P_1 \mid y(t:T).P_2)\big] \to \mathcal{E}\big[(\boldsymbol{\nu} xy : S)(P_1 \mid P_2[v/t])\big] \triangleq P'$.

   Since $\Gamma \vdash_{\mathrm{ST}} P$, by Theorem 2.2 $\Gamma \vdash_{\mathrm{ST}} M$. By inversion, $S' = !T.S$ for some S. Then, again by inversion $\Gamma = (\Gamma_1, v : T), \Gamma_2$. By Definition 5.4, $(\!|\Gamma \vdash_{\mathrm{ST}} M|\!) = A_1 \cup A_2$, where:

$$A_1 = \big\{\mathcal{F}\big[G_2 \mid (\boldsymbol{\nu} x)\big(\overline{x}(w).\big([v \leftrightarrow w] \mid P_1^*\big) \mid P_x\big)\big] : P_1^* \in (\!|\Gamma_1, x : S \vdash_{\mathrm{ST}} P_1|\!),\ P_x \in (\!|?T.\overline{S}|\!)^x,\ G_2 \in (\!|\Gamma_2|\!)\big\}$$

$$A_2 = \big\{\mathcal{H}\big[G_1 \mid (\boldsymbol{\nu} y)\big(y(t).P_2^* \mid P_y\big)\big] : P_2^* \in (\!|\Gamma_2, y : \overline{S}, t : T \vdash_{\mathrm{ST}} P_2|\!),\ P_y \in (\!|!T.S|\!)^y,\ G_1 \in (\!|\Gamma_1, v : T|\!)\big\}$$

   Also by Definition 5.4, we have that $(\!|\Gamma \vdash_{\mathrm{ST}} P'|\!) = B_1 \cup B_2$, where:

$$B_1 = \big\{\mathcal{F}\big[G_2' \mid (\boldsymbol{\nu} x)\big(Q_1 \mid P_x'\big)\big] : P_x' \in (\!|\overline{S}|\!)^x,\ Q_1 \in (\!|\Gamma_1, x : S \vdash_{\mathrm{ST}} P_1|\!),\ G_2' \in (\!|\Gamma_2, v : T|\!)\big\}$$

$$B_2 = \big\{\mathcal{H}\big[G_1' \mid (\boldsymbol{\nu} y)\big(Q_2 \mid P_y'\big)\big] : P_y' \in (\!|S|\!)^y,\ Q_2 \in (\!|\Gamma_2, y : \overline{S}, v : T \vdash_{\mathrm{ST}} P_2[^v/_t]|\!),\ G_1' \in (\!|\Gamma_1|\!)\big\}$$

   We now address two distinct sub-cases:
   (i) We show that every $Q \in A_1$ reduces to a $Q' \in B_1$;
   (ii) Similarly, we show that every $Q \in A_2$ reduces to a Q' such that $Q' \doteqdot R$ and $R \in B_2$.

   Sub-case (i). Let $Q = \mathcal{F}\big[G_2 \mid (\boldsymbol{\nu} x)\big(\overline{x}(w).\big([v \leftrightarrow w] \mid P_1^*\big) \mid P_x\big)\big]$ be an arbitrary process in A_1, with $P_x \in (\!|?T.\overline{S}|\!)^x$. By Definition 5.1 we have that

$$(\!|?T.\overline{S}|\!)^x = \big\{x(t).(Q_t \mid Q_x) : Q_t \in (\!|T|\!)^t \wedge Q_x \in (\!|\overline{S}|\!)^x\big\}$$

   We may then let $P_x = x(t).(Q_t \mid Q_x)$ where $Q_t \in (\!|T|\!)^t$ and $Q_x \in (\!|\overline{S}|\!)^x$. By applying Rules (R-ChCom) and (R-Fwd) (cf. Fig. 4) we have:

$$\begin{aligned} Q &\triangleq \mathcal{F}\big[G_2 \mid (\boldsymbol{\nu} x)\big(\overline{x}(w).\big([v \leftrightarrow w] \mid P_1^*\big) \mid P_x\big)\big] \\ &= \mathcal{F}\big[G_2 \mid (\boldsymbol{\nu} x)\big(\overline{x}(w).\big([v \leftrightarrow w] \mid P_1^*\big) \mid x(t).(Q_t \mid Q_x)\big)\big] \\ &\to \mathcal{F}\big[G_2 \mid (\boldsymbol{\nu} x)\big((\boldsymbol{\nu} w)\big([v \leftrightarrow w] \mid P_1^* \mid Q_w \mid Q_x\big)\big)\big] \\ &\hookrightarrow \mathcal{F}\big[G_2 \mid Q_v \mid (\boldsymbol{\nu} x)\big(P_1^* \mid Q_x\big)\big] \triangleq Q' \end{aligned}$$

   It is then easy to see that $Q' \in (\!|\Gamma \vdash_{\mathrm{ST}} P'|\!)$.

   Sub-case (ii). Let $Q = \mathcal{H}\big[G_1 \mid (\boldsymbol{\nu} y)\big(y(t).P_2^* \mid P_y\big)\big]$ be an arbitrary process in A_2. By Definition 5.1, since $P_y \in (\!|!T.S|\!)^y$, we may then let $P_y = \overline{y}(k).(Q_k \mid Q_y)$ where $Q_k \in (\!|\overline{T}|\!)^k$ and $Q_y \in (\!|S|\!)^y$. By applying Rule (R-ChCom) (cf. Fig. 4), we have:

$$\begin{aligned} Q &\triangleq \mathcal{H}\big[G_1 \mid (\boldsymbol{\nu} y)\big(y(t).P_2^* \mid P_y\big)\big] \\ &= \mathcal{H}\big[G_1 \mid (\boldsymbol{\nu} y)\big(y(t).P_2^* \mid \overline{y}(k).(Q_k \mid Q_y)\big)\big] \\ &\to \mathcal{H}\big[G_1 \mid (\boldsymbol{\nu} y)\big((\boldsymbol{\nu} k)(P_2^*[k/t] \mid Q_k) \mid Q_y\big)\big] \triangleq Q' \end{aligned}$$

   By Lemma B.2, $P_2^*[k/t] \in (\!|\Gamma_2, y : \overline{S}, k : T \vdash_{\mathrm{ST}} P_2[^k/_t]|\!)$. Also, by Proposition B.1 we can rewrite G_1 as $G_1 = G_1^* \mid G_v$, such that $G_1^* \in [\![\Gamma_1]\!]_\ell$ and $G_v \in [\![v : T]\!]_\ell$. Then, we can rewrite Q' as follows:

$$Q' = \mathcal{H}\big[G_1^* \mid G_v \mid (\boldsymbol{\nu} y)\big((\boldsymbol{\nu} k)(P_2^*[k/t] \mid Q_k) \mid Q_y\big)\big]$$

   It is then easy to see that $Q' \doteqdot R$, with $R \in (\!|\Gamma \vdash_{\mathrm{ST}} P'|\!)$, as wanted.

2. Case $M = \mathcal{E}\big[(\boldsymbol{\nu} xy : S')(x \triangleleft l_j.Q \mid y \triangleright \{l_i : R_i\}_{i\in I})\big] \to \mathcal{E}\big[(\boldsymbol{\nu} xy : S'')(Q \mid R_j)\big] \triangleq P'$.

   Since $\Gamma \vdash_{\mathrm{ST}} P$, by Theorem 2.2 also $\Gamma \vdash_{\mathrm{ST}} M$. By inversion let $S' = \oplus\{l_i : S_i\}_{i\in I}$ for some session types S_i $(i \in I)$. By Definition 5.4, $(\!|\Gamma \vdash_{\mathrm{ST}} M|\!) = A_1 \cup A_2$, where:

$$A_1 = \big\{\mathcal{F}\big[G_2 \mid (\boldsymbol{\nu} x)\big(x \triangleleft l_j.Q^* \mid P_x\big)\big] : Q^* \in (\!|\Gamma_1, x : S_j \vdash_{\mathrm{ST}} Q|\!),\ P_x \in (\!|\&\{l_i : T_i\}_{i\in I}|\!)^x,\ G_2 \in (\!|\Gamma_2|\!)\big\}$$

$$A_2 = \big\{\mathcal{H}\big[G_1 \mid (\boldsymbol{\nu} y)\big(y \triangleright \{l_i : R_i^*\}_{i\in I} \mid P_y\big)\big] : R_i^* \in (\!|\Gamma_2, y : T_i \vdash_{\mathrm{ST}} R_i|\!),\ P_y \in (\!|\oplus\{l_i : S_i\}_{i\in I}|\!)^y,\ G_1 \in (\!|\Gamma_1|\!)\big\}$$

Also, by Definition 5.4, we have that $(\!|\Gamma \vdash_{\mathrm{ST}} P'|\!) = B_1 \cup B_2$, where:

$$B_1 = \left\{ \mathcal{F}\left[G_2' \mid (\boldsymbol{\nu} x)(Q^* \mid P_x')\right] : Q^* \in (\!|\Gamma_1, x : S_j \vdash_{\mathrm{ST}} Q|\!),\ P_x' \in (\!|T_j|\!)^x,\ G_2' \in (\!|\Gamma_2|\!) \right\}$$

$$B_2 = \left\{ \mathcal{H}\left[G_1' \mid (\boldsymbol{\nu} y)(R_j^* \mid P_y')\right] : R_j^* \in (\!|\Gamma_2, y : T_j \vdash_{\mathrm{ST}} R_j|\!),\ P_y' \in (\!|S_j|\!)^y,\ G_1' \in (\!|\Gamma_1|\!) \right\}$$

As before, we now address two distinct sub-cases:
(i) We show that every $Q \in A_1$ reduces to a $Q' \in B_1$;
(ii) Similarly, we show that every $Q \in A_2$ reduces to a $Q' \in B_2$.

Sub-case (i). Let $Q = \mathcal{F}\left[G_2 \mid (\boldsymbol{\nu} x)\left(x \triangleleft l_j.Q^* \mid P_x\right)\right]$ be an arbitrary process in A_1. By Definition 5.1, since $P_x \in (\!|\&\{l_i : T_i\}_{i\in I}|\!)^x$, then $P_x = x \triangleright \{l_i : P_i\}_{i\in I}$ with $P_i \in (\!|T_i|\!)^x$, for all $i \in I$. By applying Rule (R-ChCase) (cf. Fig. 4), and letting $j \in I$ we have:

$$\begin{aligned} Q &\triangleq \mathcal{F}\left[G_2 \mid (\boldsymbol{\nu} x)\left(x \triangleleft l_j.Q^* \mid P_x\right)\right] \\ &\to \mathcal{F}\left[G_2 \mid (\boldsymbol{\nu} x)\left(Q^* \mid P_j\right)\right] \triangleq Q' \end{aligned}$$

It is then easy to see that $Q' \in B_1$, and therefore $Q' \in (\!|\Gamma \vdash_{\mathrm{ST}} P'|\!)$, as desired.

Sub-case (ii). Let $Q = \mathcal{H}\left[G_1 \mid (\boldsymbol{\nu} y)\left(y \triangleright \{l_i : R_i^*\}_{i\in I} \mid y \triangleleft l_j.P_j\right)\right]$ be an arbitrary process in A_2, with $R_i^* \in (\!|\Gamma_2, y : T_i \vdash_{\mathrm{ST}} R_i|\!)$. By Definition 5.1, since $P_y \in (\!|\oplus\{l_i : S_i\}_{i\in I}|\!)^y = \bigcup_{i\in I}\left\{y \triangleleft l_i.P_i : P_i \in (\!|S_i|\!)^y\right\}$, it means that P_y is one of the processes in the union. We may choose $P_y = y \triangleleft l_j.P_j$ such that $j \in I$ and $P_j \in (\!|S_j|\!)^y$. By applying Rule (R-ChCase) (cf. Fig. 4) we have:

$$\begin{aligned} Q &\triangleq \mathcal{H}\left[G_1 \mid (\boldsymbol{\nu} y)\left(y \triangleright \{l_i : R_i^*\}_{i\in I} \mid y \triangleleft l_j.P_j\right)\right] \\ &\to \mathcal{H}\left[G_1 \mid (\boldsymbol{\nu} y)\left(R_j^* \mid P_j\right)\right] \triangleq Q' \end{aligned}$$

Since $P_j \in (\!|S_j|\!)^y$ and $P_y' \in (\!|S_j|\!)^y$, it is then easy to see that $Q' \in B_2$, and therefore $Q' \in (\!|\Gamma \vdash_{\mathrm{ST}} P'|\!)$, as desired. This concludes the analysis for this case (and for Part 2). □

References

[1] Samson Abramsky, Simon J. Gay, Rajagopal Nagarajan, A type-theoretic approach to deadlock-freedom of asynchronous systems, in: Martín Abadi, Takayasu Ito (Eds.), Theoretical Aspects of Computer Software, Proceedings of the Third International Symposium, TACS '97, Sendai, Japan, September 23–26, 1997, in: Lecture Notes in Computer Science, vol. 1281, Springer, 1997, pp. 295–320.

[2] Robert Atkey, Sam Lindley, J. Garrett Morris, Conflation confers concurrency, in: Sam Lindley, Conor McBride, Philip W. Trinder, Donald Sannella (Eds.), A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday, in: Lecture Notes in Computer Science, vol. 9600, Springer, 2016, pp. 32–55.

[3] Lorenzo Bettini, Mario Coppo, Loris D'Antoni, Marco De Luca, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, Global progress in dynamically interleaved multiparty sessions, in: CONCUR, 2008, pp. 418–433.

[4] Michele Boreale, Davide Sangiorgi, A fully abstract semantics for causality in the π-calculus, Acta Inform. 35 (5) (1998) 353–400.

[5] Stephanie Balzer, Bernardo Toninho, Frank Pfenning, Manifest deadlock-freedom for shared session types, in: Luís Caires (Ed.), Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Proceedings, Prague, Czech Republic, April 6–11, 2019, in: Lecture Notes in Computer Science, vol. 11423, Springer, 2019, pp. 611–639.

[6] Luís Caires, Types and logic, concurrency and non-determinism, in: Essays for the Luca Cardelli Fest, September 2014, Microsoft Research Technical Report MSR-TR-2014-104.

[7] Marco Carbone, Søren Debois, A graphical approach to progress for structured communication in web services, in: ICE 2010, Amsterdam, The Netherlands, 10th of June 2010, in: EPTCS, vol. 38, 2010, pp. 13–27.

[8] Marco Carbone, Ornela Dardha, Fabrizio Montesi, Progress as compositional lock-freedom, in: Coordination Models and Languages - 16th IFIP WG 6.1 International Conference, COORDINATION 2014, Held as Part of the 9th International Federated Conferences on Distributed Computing Techniques, DisCoTec, in: LNCS, vol. 8459, Springer, 2014, pp. 49–64.

[9] Luís Caires, Frank Pfenning, Session types as intuitionistic linear propositions, in: CONCUR 2010, in: LNCS, vol. 6269, Springer, 2010, pp. 222–236.

[10] Luís Caires, Jorge A. Pérez, Linearity, control effects, and behavioral types, in: Hongseok Yang (Ed.), Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Proceedings, Uppsala, Sweden, April 22–29, 2017, in: Lecture Notes in Computer Science, vol. 10201, Springer, 2017, pp. 229–259.

[11] Luís Caires, Frank Pfenning, Bernardo Toninho, Linear logic propositions as session types, Math. Struct. Comput. Sci. 26 (3) (2016) 367–423.

[12] Luís Caires, Hugo Torres Vieira, Conversation types, in: ESOP'09, in: LNCS, vol. 5502, Springer-Verlag, Heidelberg, Germany, 2009, pp. 285–300.

[13] Ornela Dardha, Recursive session types revisited, in: Proceedings Third Workshop on Behavioural Types, BEAT 2014, Rome, Italy, 1st September 2014, in: EPTCS, vol. 162, 2014, pp. 27–34.

[14] Ornela Dardha, Type Systems for Distributed Programs: Components and Sessions, Atlantis Studies in Computing, vol. 7, Springer/Atlantis Press, 2016.

[15] Mariangiola Dezani-Ciancaglini, Ugo de'Liguoro, Nobuko Yoshida, On progress for structured communications, in: Trustworthy Global Computing, in: LNCS, vol. 4912, Springer, 2008, pp. 257–275.

[16] Ornela Dardha, Simon J. Gay, 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, Proceedings, Thessaloniki, Greece, April 14–20, 2018, in: LNCS, vol. 10803, Springer, 2018, pp. 91–109.

[17] Ornela Dardha, Elena Giachino, Davide Sangiorgi, Session types revisited, in: PPDP'12, ACM, 2012, pp. 139–150.

[18] Ornela Dardha, Elena Giachino, Davide Sangiorgi, Session types revisited, Inf. Comput. 256 (2017) 253–286.

[19] Ornela Dardha, Jorge A. Pérez, Comparing deadlock-free session typed processes, in: Combined 22nd International Workshop on Expressiveness in Concurrency and 12th Workshop on Structural Operational Semantics, EXPRESS/SOS, in: EPTCS, vol. 190, 2015, pp. 1–15.
[20] Ornela Dardha, Jorge A. Pérez, Comparing type systems for deadlock freedom, CoRR, arXiv:1810.00635, 2020.
[21] Adrian Francalanza, Marco Giunti, António Ravara, Unlocking blocked communicating processes, in: Maurice H. ter Beek, Alberto Lluch-Lafuente (Eds.), Proceedings 11th International Workshop on Automated Specification and Verification of Web Systems, WWV 2015, Oslo, Norway, 23rd June 2015, in: EPTCS, vol. 188, 2015, pp. 23–32.
[22] Simon Fowler, Wen Kokke, Ornela Dardha, Sam Lindley, J. Garrett Morris, Separating sessions smoothly, in: Serge Haddad, Daniele Varacca (Eds.), 32nd International Conference on Concurrency Theory, CONCUR 2021, Virtual Conference, August 24–27, 2021, in: LIPIcs, vol. 203, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 36.
[23] Simon Gay, Malcolm Hole, Subtyping for session types in the pi calculus, Acta Inform. 42 (2–3) (Nov 2005) 191–225.
[24] Elena Giachino, Naoki Kobayashi, Cosimo Laneve, Deadlock analysis of unbounded process networks, in: CONCUR, 2014, pp. 63–77.
[25] Daniele Gorla, Uwe Nestmann, Full abstraction for expressiveness: history, myths and facts, Math. Struct. Comput. Sci. 26 (4) (2016) 639–654.
[26] Daniele Gorla, Towards a unified approach to encodability and separation results for process calculi, Inf. Comput. 208 (9) (2010) 1031–1053.
[27] Marco Giunti, António Ravara, Towards static deadlock resolution in the π-calculus, in: TGC, 2013, pp. 136–155.
[28] Kohei Honda, Types for dyadic interaction, in: The 4th International Conference on Concurrency Theory, CONCUR, in: LNCS, vol. 715, Springer, 1993, pp. 509–523.
[29] Kohei Honda, Vasco Thudichum Vasconcelos, Makoto Kubo, Language primitives and type discipline for structured communication-based programming, in: ESOP'98, in: LNCS, vol. 1381, Springer, 1998, pp. 122–138.
[30] Atsushi Igarashi, Naoki Kobayashi, A generic type system for the pi-calculus, Theor. Comput. Sci. 311 (1–3) (2004) 121–163.
[31] Wen Kokke, Ornela Dardha, Deadlock-free session types in linear Haskell, in: Jurriaan Hage (Ed.), Haskell 2021: Proceedings of the 14th ACM SIGPLAN International Symposium on Haskell, Virtual Event, Korea, August 26–27, 2021, ACM, 2021, pp. 1–13.
[32] Wen Kokke, Ornela Dardha, Prioritise the best variation, in: Kirstin Peters, Tim A.C. Willemse (Eds.), Formal Techniques for Distributed Objects, Components, and Systems - 41st IFIP WG 6.1 International Conference, FORTE 2021, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques, DisCoTec 2021, Proceedings, Valletta, Malta, June 14–18, 2021, in: Lecture Notes in Computer Science, vol. 12719, Springer, 2021, pp. 100–119.
[33] Naoki Kobayashi, Cosimo Laneve, Deadlock analysis of unbounded process networks, Inf. Comput. 252 (2017) 48–70.
[34] Wen Kokke, Fabrizio Montesi, Marco Peressotti, Better late than never: a fully-abstract semantics for classical processes, Proc. ACM Program. Lang. 3 (POPL) (2019) 24.
[35] Naoki Kobayashi, A partially deadlock-free typed process calculus, in: Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 – July 2, 1997, IEEE Computer Society, 1997, pp. 128–139.
[36] Naoki Kobayashi, A type system for lock-free processes, Inf. Comput. 177 (2) (2002) 122–159.
[37] Naoki Kobayashi, Type systems for concurrent programs, in: Formal Methods at the Crossroads: From Panacea to Foundational Support—Papers from the 10th Anniversary Colloquium of UNU/IIST the International Institute for Software Technology of the United Nations University, in: LNCS, vol. 2757, Springer, 2002, pp. 439–453.
[38] Naoki Kobayashi, A new type system for deadlock-free processes, in: CONCUR 2006, in: LNCS, vol. 4137, Springer, 2006, pp. 233–247, full version available at http://www-kb.is.s.u-tokyo.ac.jp/~koba/papers/concur2006-full.pdf.
[39] Naoki Kobayashi, Type systems for concurrent programs, Extended version of [37], Tohoku University, http://www.kb.ecei.tohoku.ac.jp/~koba/papers/tutorial-type-extended.pdf, 2007.
[40] Naoki Kobayashi, Benjamin C. Pierce, David N. Turner, Linearity and the pi-calculus, in: POPL, 1996, pp. 358–371.
[41] Naoki Kobayashi, Benjamin C. Pierce, David N. Turner, Linearity and the pi-calculus, ACM Trans. Program. Lang. Syst. 21 (5) (1999) 914–947.
[42] Naoki Kobayashi, Davide Sangiorgi, A hybrid type system for lock-freedom of mobile processes, ACM Trans. Program. Lang. Syst. 32 (5) (2010).
[43] Sam Lindley, J. Garrett Morris, Talking bananas: structural recursion for session types, in: Jacques Garrigue, Gabriele Keller, Eijiro Sumii (Eds.), Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18–22, 2016, ACM, 2016, pp. 434–447.
[44] Robin Milner, The Polyadic pi-Calculus: A Tutorial, Technical report, Logic and Algebra of Specification, 1991.
[45] Robin Milner, Joachim Parrow, David Walker, A calculus of mobile processes, I, Inf. Comput. 100 (1) (1992) 1–40.
[46] Luca Padovani, From lock freedom to progress using session types, in: Proceedings of PLACES 2013, Rome, Italy, 23rd March 2013, in: EPTCS, vol. 137, 2013, pp. 3–19.
[47] Luca Padovani, Deadlock and lock freedom in the linear π-calculus, in: CSL-LICS, ACM, 2014, 72.
[48] Joachim Parrow, Expressiveness of process algebras, Electron. Notes Theor. Comput. Sci. 209 (2008) 173–186.
[49] Joachim Parrow, General conditions for full abstraction, Math. Struct. Comput. Sci. 26 (4) (2016) 655–657.
[50] Jorge A. Pérez, Luís Caires, Frank Pfenning, Bernardo Toninho, Linear logical relations and observational equivalences for session-based concurrency, Inf. Comput. 239 (2014) 254–302.
[51] Benjamin C. Pierce, Types and Programming Languages, MIT Press, MA, USA, 2002.
[52] Zesen Qian, G.A. Kavvos, Lars Birkedal, Client-server sessions in linear logic, Proc. ACM Program. Lang. 5 (ICFP) (2021) 1–31.
[53] Sangiorgi Davide, Types, or: where's the difference between CCS and pi?, in: Lubos Brim, Petr Jancar, Mojmír Kretínský, Antonín Kucera (Eds.), CONCUR 2002 - Concurrency Theory, 13th International Conference, Proceedings, Brno, Czech Republic, August 20–23, 2002, in: Lecture Notes in Computer Science, vol. 2421, Springer, 2002, pp. 76–97.
[54] Davide Sangiorgi, David Walker, The π-calculus: A Theory of Mobile Processes, Cambridge University Press, 2001.
[55] Bernardo Toninho, Luís Caires, Frank Pfenning, Corecursion and non-divergence in session-typed processes, in: TGC 2014, in: LNCS, vol. 8902, Springer, 2014, pp. 159–175.
[56] Kaku Takeuchi, Kohei Honda, Makoto Kubo, An interaction-based language and its typing system, in: PARLE'94, 1994, pp. 398–413.
[57] Vasco T. Vasconcelos, Fundamentals of session types, Inf. Comput. 217 (2012) 52–70.
[58] Bas van den Heuvel, Jorge A. Pérez, Session type systems based on linear logic: classical versus intuitionistic, in: Stephanie Balzer, Luca Padovani (Eds.), Proceedings of the 12th International Workshop on Programming Language Approaches to Concurrency- and Communication-cEntric Software, PLACES@ETAPS 2020, Dublin, Ireland, 26th April 2020, in: EPTCS, vol. 314, 2020, pp. 1–11.
[59] Bas van den Heuvel, Jorge A. Pérez, Deadlock freedom for asynchronous and cyclic process networks, in: Julien Lange, Anastasia Mavridou, Larisa Safina, Alceste Scalas (Eds.), Proceedings 14th Interaction and Concurrency Experience, ICE 2021, in: EPTCS, 2021.
[60] Hugo Torres Vieira, Vasco Thudichum Vasconcelos, Typing progress in communication-centred systems, in: COORDINATION, in: LNCS, vol. 7890, Springer, 2013, pp. 236–250.
[61] Philip Wadler, Propositions as sessions, in: ICFP'12, 2012, pp. 273–286.
[62] Nobuko Yoshida, Graph types for monadic mobile processes, in: FSTTCS'96, in: LNCS, vol. 1180, Springer, 1996, pp. 371–386.