Contrasting Deadlock-Free Session Processes

Juan C. Jaramillo
Unversity of Groningen, The Netherlands

Jorge A. Pérez
Unversity of Groningen, The Netherlands

Abstract

Deadlock freedom is a crucial property for message-passing programs. Over the years, several different type systems for concurrent processes that ensure deadlock freedom have been proposed; this diversity raises the question of how they compare. We address this question, considering two type systems not covered in prior work: Kokke *et al.*'s HCP, a type system based on a linear logic with *hypersequents*, and Padovani's *priority-based* type system for asynchronous processes, dubbed P. Their distinctive features make formal comparisons relevant and challenging. Our findings are two-fold: (1) the hypersequent setting does not drastically change the class of deadlock-free processes induced by linear logic, and (2) we relate the classes of deadlock-free processes induced by HCP and P. We prove that our results hold under both synchronous and asynchronous communication. Our results provide new insights into the essential mechanisms involved in statically avoiding deadlocks in concurrency.

2012 ACM Subject Classification Theory of computation → Type structures; Theory of computation → Program analysis; Theory of computation → Process calculi

Keywords and phrases session types, process calculi, deadlock freedom

Digital Object Identifier 10.4230/LIPIcs.ECOOP.2025.17

Related Version *Full Version*: `https://arxiv.org/abs/2504.15845` [19]

Funding *Juan C. Jaramillo*: Ministry of Science of Colombia (Minciencias).

Acknowledgements We are grateful to Channa Dias Perera for initial discussions on the topics of this paper. We also thank Andreia Mordido and the anonymous reviewers for their helpful suggestions.

1 Introduction

The *deadlock freedom* property (DF, in the following) is a crucial guarantee for message-passing programs. Informally, DF ensures that processes "never get stuck" – an essential requirement in concurrent and distributed software systems, where even a single component that becomes permanently blocked while waiting for a message can compromise reliability. As illustration, consider a toy concurrent program in the language of Gay and Vasconcelos [14]:

$$\langle \mathsf{let}\,(x_1, z) = \mathsf{receive}\, x_1 \;\; \mathsf{in}\;\, \mathsf{send}\;\, 42\, y_1\rangle \parallel \langle \mathsf{let}\,(y_2, k) = \mathsf{receive}\, y_2 \;\; \mathsf{in}\;\, \mathsf{send}\;\, \text{`}\mathtt{hello}\text{'}\, x_2\rangle$$

We have two concurrent threads, each running an expression with two actions in sequence. This program is evaluated under a context that binds together x_1 and x_2 and declares them as endpoints of the same linear channel (same for y_1 and y_2); we omit it for readability. Given a linear channel x, "$\mathsf{send}\;\mathtt{v}\,x$" sends value $\mathtt{v}$ along x and returns the continuation of x, whereas "$\mathsf{receive}\,x$" returns a pair with the continuation of x and a received value. The threads are thus engaged in separate but intertwined protocols (one along x_i, the other along y_i). Each thread should receive on a channel and then send a value on the other. However, the output matching the input in one thread is blocked in the other – the program is stuck. The key challenge in enforcing DF is thus performing the non-local analysis needed to avoid *circular dependencies* between threads, which are often less apparent than in this program.

An expressive model for concurrent processes, the π-calculus offers a convenient basis for developing analysis techniques for message-passing programs in different paradigms [32]. Several type systems that enforce (forms of) DF have been proposed over the years (see, e.g., [21, 11, 5, 27, 7, 34, 1]). They avoid circular dependencies in processes using different insights, which raises the question of how they compare. In this paper, we rigorously compare some representative type systems for DF to better understand their underlying mechanisms.

A key motivation for studying type systems for concurrent processes is their suitability for developing advanced analysis techniques at an appropriate level of abstraction – these typed frameworks have been directly incorporated into languages such as Scala [33], OCaml [28, 17], and Rust [24]. Also, concurrent processes offer an adequate representation level for verifying liveness properties in real Go programs [25, 26]. Moreover, type systems for the π-calculus are the basis for static analyzers (such as Kobayashi's TyPiCal [20]) and can be ported to enforce DF for higher-order concurrent programs [29]. Type systems based on the Curry-Howard correspondence between session types and linear logic ("propositions-as-sessions" [3, 37]) enforce DF; they entail topological invariants on processes, key to ensure deadlock and memory leak freedom in concurrent programs [18]. Further understanding the essential mechanisms that enforce DF in logic-based session types is a central topic in our work.

Prior work by Dardha and Pérez [9] contrasts type systems for DF by relating the classes of processes they induce. They compare a *priority-based* type system by Kobayashi [21] and a type system derived from "propositions-as-sessions". Writing $\mathbb{K}$ and $\mathbb{L}$ to denote the respective classes of processes, a result in [9] is that $\mathbb{L} \subsetneq \mathbb{K}$: there exist deadlock-free processes that cannot be typed by logic-based type systems. The class $\mathbb{K} \setminus \mathbb{L}$ includes, for instance, process networks organized in circular topologies (e.g., the dinning philosophers). It also includes process P_0 below, in which two processes interact along independent protocols (*sessions*):

$$P_0 \triangleq (\nu\, x_1 x_2)(\nu\, y_1 y_2)(x_1(z).\overline{y_1}\langle \texttt{42}\rangle.\mathbf{0} \mid \overline{x_2}\langle \texttt{hello}\rangle.y_2(k).\mathbf{0})$$

Above, "$(\nu\, x_1 x_2)$" declares a new session with linear endpoints x_1 and x_2, and "$x(y).Q$" (resp. "$\overline{x}\langle v\rangle.Q$") denotes an input (resp. output of value v) along x with continuation Q. P_0 is the deadlock-free variant of the program given above, now neatly specified as a process. In $\mathbb{L}$, processes can share at most one session in parallel, whereas processes in P_0 share two: the session "$x_1 x_2$" for exchanging "hello" and the session "$y_1 y_2$" for exchanging "42". In [9], both compared classes rely on a reduction semantics; accordingly, DF concerns the absence of stable states with *pending reduction steps* on linear names/channels.

Here we extend the work in [9] by considering type systems not studied there: (i) Kokke *et al.*'s HCP [23] and (ii) Padovani's priority-based type system [27], here dubbed P. Their distinctive features make formal comparisons relevant and challenging, as we discuss next.

Derived from "propositions-as-sessions", HCP is based on a presentation of classical linear logic (CLL) with *hypersequents*. HCP has been studied in several works, which have deepened into its theory and applications [22, 12, 31]. In HCP, the semantics of processes is given by a Labeled Transition System (LTS), rather than by a reduction semantics. This unlocks the use of well-established analysis techniques, notably observational equivalences, to reason about typed processes. The (typed) LTS for HCP includes the usual rules for observing actions and synchronizations, here dubbed *regular transitions*, but also rules that allow for *delayed actions* (i.e., observable actions due to prefixes not at top-level) and *self-synchronizations* (i.e., internal actions that do not require the interaction of two parallel processes). Remarkably, the bisimilarity induced by the LTS coincides, in a fully-abstract way, with a *denotational equivalence* over processes – this gives a strong justification for the design of the LTS. Typing ensures *progress*: well-typed processes not equivalent to $\mathbf{0}$ always have a transition step.

An intriguing observation is that there are HCP processes that are stuck in a reduction-based setting but have synchronizations under HCP's LTS. We discuss two interesting examples, using "$x().Q$" and "$\overline{x}\langle\,\rangle.Q$" to denote empty inputs and outputs, respectively:

- Process $P_1 \triangleq (\nu\, xx')\overline{x'}\langle\,\rangle.x().\mathbf{0}$ is self-synchronizing: it does not have any reductions, whereas $P_1 \xrightarrow{\tau} \mathbf{0}$, as in HCP synchronizations do not need a parallel context to be enabled.
- Process $P_2 \triangleq (\nu\, xx')(\nu\, yy')(\overline{y}\langle\,\rangle.x().\mathbf{0} \mid \overline{x'}\langle\,\rangle.y'().\mathbf{0})$ is arguably the paradigmatic example of a deadlock, with two independent sessions unable to reduce. In HCP, thanks to delayed actions, we have both $P_2 \xrightarrow{\tau} (\nu\, yy')(\overline{y}\langle\,\rangle.\mathbf{0} \mid y'().\mathbf{0})$ and $P_2 \xrightarrow{\tau} (\nu\, xx')(\overline{x}\langle\,\rangle.\mathbf{0} \mid x'().\mathbf{0})$.

This observation shows that comparisons such as [9] should accommodate other semantics for processes, beyond reductions, which entail other definitions of DF and induce new deadlock-free processes (such as P_1 and P_2). This begs a first question:

(Q1) What is the status of delayed actions and self-synchronizations, as typable in HCP, with respect to DF? Do they induce a distinct class of deadlock-free processes?

Considering Padovani's P means addressing the case of processes with *asynchronous communication*, widely used in real programs. In the π-calculus, asynchronous communication is obtained by defining outputs as non-blocking actions; only inputs can block the execution of a process [2, 16]. Asynchrony and DF are then closely related: because an asynchronous setting involves the least amount of blocking operations, it offers the most elementary scenario for investigating DF [35]. Now, the comparisons in [9] are limited to synchronous processes; we then observe that they should be extended to address asynchronous communication, too. Let us write $\mathbb{H}$ and $\mathbb{P}$ to denote the class of processes induced by HCP and P, respectively, and $\mathbb{C}$ to denote the class of processes induced by CP (a fragment of HCP without hypersequents and with a restricted form of parallel composition). We consider the second question:

(Q2) What is the status of $\mathbb{H}$ and $\mathbb{C}$ with respect to asynchronous processes in $\mathbb{P}$?

Interestingly, our two questions are intertwined: HCP and P are related due to asynchrony. Our insight is that delayed actions in HCP already implement an asynchronous semantics. To see this, consider the process $P_3 \triangleq (\nu\, xx')(\nu\, yy')(\overline{y}\langle\,\rangle.\overline{x}\langle\,\rangle.\mathbf{0} \mid x'().y'().\mathbf{0})$, which is not stuck: the output on y is not really blocking, as delayed actions allow to anticipate the output (and synchronization) on x. In other words, P_3 is morally the same as $(\nu\, xx')(\nu\, yy')(\overline{y}\langle\,\rangle.\mathbf{0} \mid \overline{x}\langle\,\rangle.\mathbf{0} \mid x'().y'().\mathbf{0})$. We thus conclude that the (typed) operational semantics for HCP admits many synchronous-looking processes (such as P_3) that actually behave in an asynchronous way; this naturally invites the question of how $\mathbb{H}$ relates to $\mathbb{P}$.

Contributions and Outline. This paper offers conceptual and technical contributions. From a conceptual view, we reflect upon the significance of formally comparing different type systems for concurrency, while uniformly presenting various notions in the literature. On the technical side, we address **(Q1)** and **(Q2)**, thus extending and complementing [9]. A key discovery is identifying *commuting conversions* and *disentanglement* as key notions for characterizing deadlock-free processes in HCP – as we will see, these notions define sound *process optimizations* that respect causality and increase parallelism. As our results hold also in an asynchronous setting, not treated in prior work, our findings are most related to abstractions and mechanisms used in practical concurrent programming. More in detail:

§ 2 We introduce SP, a session π-calculus that we use as baseline for comparisons, as well as CP and HCP, the type systems resulting from the Curry-Howard correspondence with CLL and CLL with hypersequents, respectively.

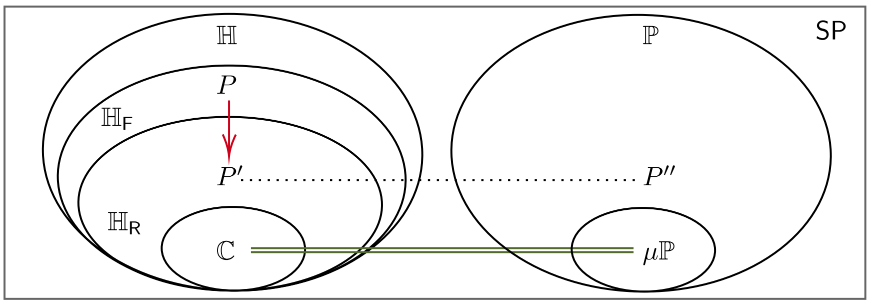


Figure 1 Overview of main results. Above, $\mathbb{H}_\mathsf{F}$, $\mathbb{H}_\mathsf{R}$, $\mathbb{P}$, $\mathbb{C}$, and $\mu\mathbb{P}$ stand for classes of deadlock-free SP processes. The red arrow represents optimizations (commuting conversions and disentanglement) that relate $\mathbb{H}_\mathsf{F}$ and $\mathbb{H}_\mathsf{R}$. The green lines denote the correspondence $\mathbb{C} = \mu\mathbb{P}$. The dotted line means that P' and P'' are the representatives of P in $\mathbb{H}_\mathsf{R}$ and $\mathbb{P}$, up to an encoding.

§ 3 We define $\mathbb{C}$ and $\mathbb{H}$: the classes of deadlock-free SP processes induced by CP and HCP, respectively. We identify two relevant sub-classes of $\mathbb{H}$:
$\mathbb{H}_\mathsf{F}$**:** Processes that use the Full LTS.
$\mathbb{H}_\mathsf{R}$**:** Processes that *only* use Regular transitions (no delayed actions/self-synchronizations).
In our main result (Corollary 53), we prove that for every process $P \in \mathbb{H}_\mathsf{F}$, there exists an observationally equivalent process $P' \in \mathbb{H}_\mathsf{R}$. Enabled by process optimizations induced via commuting conversions and disentanglement, this result shows that delayed actions and self-synchronizations are *inessential* when it comes to DF, addressing **(Q1)**.

§ 4 To address **(Q2)**, we define $\mathbb{P}$: the class of deadlock free SP processes induced by P, and its sub-class $\mu\mathbb{P}$ ("micro $\mathbb{P}$"), which is inspired by CP. We prove that $\mathbb{C} = \mu\mathbb{P}$ (Corollary 74), thus casting Dardha and Pérez's key finding into the asynchronous setting. Our main results are: Theorem 76, which precisely relates the classes $\mathbb{H}$, $\mathbb{H}_\mathsf{F}$, $\mathbb{H}_\mathsf{R}$, and $\mathbb{P}$; and Corollary 77, which strengthens Corollary 53 by considering DF in $\mathbb{P}$. Moreover, we introduce SP^a, an asynchronous variant of SP, and its corresponding classes of processes. We show how to transfer Theorem 76 and Corollary 77 from SP to SP^a (Corollary 88).

Figure 1 offers a graphical description of the classes of processes and some of our main results (Corollaries 53 and 74 and Theorem 76). For simplicity, the figure does not depict several encodings needed to bridge syntactic differences between the various classes of processes / type systems under consideration; Figure 12 (Page 26) presents a detailed version. In § 5 we collect some closing remarks. The extended version of the paper [19] contains omitted proofs.

2 Session Processes and Their Type Systems

We present SP, our reference language of session processes. As observed by Dardha and Pérez [9], this is a convenient framework for formal comparisons, because typing in SP ensures communication safety but not DF. We also present CP and HCP, the two typed languages based on "propositions-as-sessions", and recall useful associated results.

2.1 Session Processes (SP)

SP is a core session-typed π-calculus, based on the language defined by Vasconcelos [36]. We assume a base set of *variables*, ranged over by $x, y, \ldots$, which denote *channels* (or *names*): they can be seen as *endpoints* on which interaction takes place. Processes interact to exchange values $v, v', \ldots$; for simplicity, values correspond to variables, i.e., $v ::= x$. We write $\widetilde{x}$ to denote a finite sequence of variables. The syntax of *processes* is as follows:

$$P \quad ::= \quad \mathbf{0} \mid \overline{x}\langle v\rangle.P \mid x(y).P \mid P_1 \mid P_2 \mid (\nu\, xy)P \mid \overline{x}\langle\,\rangle.P \mid x().P$$

$$(\nu\, xy)(\overline{x}\langle v\rangle.P \mid y(z).Q \mid S) \longrightarrow (\nu\, xy)(P \mid Q\{v/z\} \mid S) \qquad \text{R-Com}$$

$$(\nu\, xy)(\overline{x}\langle\,\rangle.P \mid y().Q) \longrightarrow P \mid Q \qquad \text{R-EmptyCom}$$

$$\text{R-Par}\ \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \qquad \text{R-Res}\ \frac{P \longrightarrow P'}{(\nu\, xy)P \longrightarrow (\nu\, xy)P'} \qquad \text{R-Struct}\ \frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q}$$

$$\begin{array}{rcl@{\qquad}rcl}
P \mid Q & \equiv & Q \mid P & P \mid \mathbf{0} & \equiv & P \\
P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R & (\nu\, xy)P & \equiv & (\nu\, yx)P \\
(\nu\, xy)P \mid Q & \equiv & (\nu\, xy)(P \mid Q) \ \text{ if } x, y \notin \texttt{fv}(Q) & (\nu\, xy)(\nu\, wz)P & \equiv & (\nu\, wz)(\nu\, xy)P
\end{array}$$

Figure 2 SP: Reduction semantics (top) and structural congruence (bottom).

Process **0** denotes inaction. Process $\overline{x}\langle v\rangle.P$ sends v along x and continues as P. Process $x(y).P$ receives v along x and continues as $P\{v/y\}$, i.e., the process resulting from the capture-avoiding substitution of y by v in P. Process $P_1 \mid P_2$ denotes the parallel execution of P_1 and P_2. Process $(\nu\, xy)P$ declares x and y as *co-variables*, restricting their scope to P. That is, restriction declares co-variables as dual ends of a channel. Finally, $\overline{x}\langle\,\rangle.P$ and $x().P$ denote empty output and input (close and wait), which represent the closing of session x.

As usual, $x(y).P$ binds y in P and $(\nu\, xy)P$ binds x, y in P. The sets of free and bound variables of P, denoted $\texttt{fv}(P)$ and $\texttt{bv}(P)$, are defined accordingly. The semantics of processes is given using a reduction relation, defined next.

▶ **Definition 1** (SP: Reduction). Reduction *for* SP*, denoted* $\longrightarrow$*, is defined by the rules in Figure 2 (top). It relies on* structural congruence*, denoted* $\equiv$*, the smallest congruence on processes that satisfies the axioms in Figure 2 (bottom).*

We now define session types, which express protocols associated to channels:

▶ **Definition 2** (SP: Session Types and Typing Environments). *The syntax of session types and typing environments is inductively defined as follows:*

$$T, S \ ::= \ ?S.T \mid !S.T \mid \mathsf{end}_? \mid \mathsf{end}_! \qquad\qquad \Gamma \ ::= \ \emptyset \mid \Gamma, x : T$$

The type $!S.T$ (resp. $?S.T$) is assigned to a channel able to send (resp. receive) a value of type S and to continue as T. For convenience, we have two different types for channels with terminated protocols, denoted $\mathsf{end}_?$ and $\mathsf{end}_!$. Duality on types ensures that the endpoints of a channel agree on the protocol they describe:

▶ **Definition 3** (SP: Duality). *The dual of S, denoted $\overline{S}$, is defined as follows:*

$$\overline{?S.T} \triangleq !S.\overline{T} \qquad \overline{!S.T} \triangleq ?S.\overline{T} \qquad \overline{\mathsf{end}_?} \triangleq \mathsf{end}_! \qquad \overline{\mathsf{end}_!} \triangleq \mathsf{end}_?$$

▶ **Definition 4** (SP: Typing Judgment and Typing Rules). *The judgment $\Gamma \vdash_{\mathsf{S}} P$ describes a well-typed process under the typing environment Γ. The typing rules for* SP *are shown in Figure 3. We shall write $P \in$ SP if $\Gamma \vdash_{\mathsf{S}} P$, for some Γ.*

We now state the properties of well-typed SP processes, whose proofs follow [36]:

$$\textsc{Inact}\ \frac{}{\cdot \vdash_{\mathsf{S}} \mathbf{0}} \qquad \textsc{EInput}\ \frac{\Gamma \vdash_{\mathsf{S}} P}{\Gamma, x : \mathsf{end}_? \vdash_{\mathsf{S}} x().P} \qquad \textsc{EOutput}\ \frac{\Gamma \vdash_{\mathsf{S}} P}{\Gamma, x : \mathsf{end}_! \vdash_{\mathsf{S}} \overline{x}\langle\,\rangle.P} \qquad \textsc{Par}\ \frac{\Gamma_1 \vdash_{\mathsf{S}} P \qquad \Gamma_2 \vdash_{\mathsf{S}} Q}{\Gamma_1, \Gamma_2 \vdash_{\mathsf{S}} P \mid Q}$$

$$\textsc{Res}\ \frac{\Gamma, x : T, y : \overline{T} \vdash_{\mathsf{S}} P}{\Gamma \vdash_{\mathsf{S}} (\nu\, xy)P} \qquad \textsc{Out}\ \frac{\Gamma, x : U \vdash_{\mathsf{S}} P}{\Gamma, x : !T.U, v : T \vdash_{\mathsf{S}} \overline{x}\langle v\rangle.P} \qquad \textsc{In}\ \frac{\Gamma, y : T, x : U \vdash_{\mathsf{S}} P}{\Gamma, x : ?T.U \vdash_{\mathsf{S}} x(y).P}$$

Figure 3 SP: Typing Rules for Processes.

▶ **Theorem 5** (SP: Preservation [36]). *Suppose $\Gamma \vdash_{\mathsf{S}} P$. If $P \equiv Q$ then $\Gamma \vdash_{\mathsf{S}} Q$. Also, if $P \longrightarrow Q$ then $\Gamma \vdash_{\mathsf{S}} Q$.*

We will be interested in *well-formed processes*, defined using two auxiliary notions. First, we say that P is *prefixed at* x, written $\mathtt{Pr}(P) = x$, if P has one of the following forms: $x().P'$, $\overline{x}\langle\,\rangle.P$, $x(z).P'$, or $\overline{x}\langle v\rangle.P'$. Second, we say a process $(\nu\, xy)(P \mid Q)$ is a *redex* if: (1) $P = x().P'$ and $Q = \overline{y}\langle\,\rangle.Q'$, or (2) $P = x(z).P'$ and $Q = \overline{y}\langle v\rangle.Q'$. We may now define:

▶ **Definition 6** (SP: Well-formedness). *A process is well-formed if for each of its structural congruent processes of the form $(\nu\, x_1y_1)\ldots(\nu\, x_ny_n)(P_1 \mid \cdots \mid P_m)$ where $\mathtt{Pr}(P_i) = x_k$ and $\mathtt{Pr}(P_j) = y_k$, then $(\nu\, x_ky_k)(P_i \mid P_j)$ is a redex.*

▶ **Theorem 7** (Safety [36]). *Typable processes are well-formed.*

As a consequence of Theorems 5 and 7 we obtain the main guarantee of SP: *typable processes only reduce to well-formed processes.*

▶ **Corollary 8** ([36]). *If $\Gamma \vdash_{\mathsf{S}} P$ and $P \longrightarrow^* Q$, then Q is well-formed.*

Even though typability ensures well-formedness, which excludes error processes such as $(\nu\, xy)(\overline{x}\langle v\rangle.P \mid y().Q)$ and $(\nu\, xy)(x().P \mid y().Q)$, typability in SP also admits well-typed process that exhibit unwanted behaviors, in particular deadlocks. Before illustrating this fact, we give the first definition of deadlock freedom that we will encounter, which follows [9]. We write $(\nu\, \widetilde{xy})Q$ to stand for $(\nu\, x_1y_1)\cdots(\nu\, x_ny_n)Q$, for some $n \geq 0$.

▶ **Definition 9** (SP: Deadlock Freedom). *A process $P \in$ SP is deadlock free, written $\mathcal{D}_{\mathbf{S}}(P)$, if the following condition holds: whenever $P \longrightarrow^* P'$ and one of the following holds: (1) $P' \equiv (\nu\, \widetilde{xy})(\overline{x_i}\langle v\rangle.Q_1 \mid Q_2)$, (2) $P' \equiv (\nu\, \widetilde{xy})(x_i(z).Q_1 \mid Q_2)$, (3) $P' \equiv (\nu\, \widetilde{xy})(x_i().Q_1 \mid Q_2)$, or (4) $P' \equiv (\nu\, \widetilde{xy})(\overline{x_i}\langle\,\rangle.Q_1 \mid Q_2)$ (with $x_i \in \widetilde{x}$ in all cases), then $P' \longrightarrow R$, for some R.*

Hence, $\mathcal{D}_{\mathbf{S}}(P)$ is read as "P is deadlock-free in SP". Later on, we will encounter analogous notations for HCP and P; they will be denoted $\mathcal{D}_{\mathbf{H}}(P)$ and $\mathcal{D}_{\mathbf{P}}(P)$, respectively.

▶ **Example 10** (SP: A Typable but Deadlocked Process). Consider the process P_{10} defined as

$$P_{10} \triangleq (\nu\, x_1y_1)(\nu\, x_2y_2)(\nu\, v_1k_1)(\nu\, v_2k_2)(x_1(z_1).x_2(z_2).C_1 \mid \overline{y_2}\langle v_2\rangle.\overline{y_1}\langle v_1\rangle.C_2)$$

where $C_1 \triangleq x_1().x_2().z_1().z_2().\mathbf{0}$ and $C_2 \triangleq \overline{y_1}\langle\,\rangle.\overline{y_2}\langle\,\rangle.\overline{k_1}\langle\,\rangle.\overline{k_2}\langle\,\rangle.\mathbf{0}$. Similar to P_2 (§ 1), the structure of actions in process P_{10} induces a circular dependency that prevents synchronizations: we have that $\neg\mathcal{D}_{\mathbf{S}}(P_{10})$ and yet P_{10} is well-typed with the empty environment.

$$\textsc{C-}\mathbf{1}\ \frac{}{x[\,].\mathbf{0} \vdash_{\mathtt{C}} x : \mathbf{1}} \qquad \textsc{C-}\bot\ \frac{P \vdash_{\mathtt{C}} \Gamma}{x().P \vdash_{\mathtt{C}} \Gamma, x : \bot} \qquad \textsc{C-Mix}_0\ \frac{}{\mathbf{0} \vdash_{\mathtt{C}} \cdot} \qquad \textsc{C-}\otimes\ \frac{P \vdash_{\mathtt{C}} \Gamma, y : A \qquad Q \vdash_{\mathtt{C}} \Delta, x : B}{x[y].(P \mid Q) \vdash_{\mathtt{C}} \Gamma, \Delta, x : A \otimes B}$$

$$\textsc{C-}\parr\ \frac{P \vdash_{\mathtt{C}} \Gamma, y : A, x : B}{x(y).P \vdash_{\mathtt{C}} \Gamma, x : A \parr B} \qquad \textsc{C-Cut}\ \frac{P \vdash_{\mathtt{C}} \Gamma, x : A \qquad Q \vdash_{\mathtt{C}} \Delta, y : A^{\perp}}{(\nu xy)(P \mid Q) \vdash_{\mathtt{C}} \Gamma, \Delta} \qquad \textsc{C-Id}\ \frac{}{[x \leftrightarrow y] \vdash_{\mathtt{C}} x : A, y : A^{\perp}}$$

Figure 4 CP: Typing rules for processes.

2.2 CP and HCP

The languages CP and HCP use the same syntax. Here we consider HCP as presented by Kokke *et al.* [23] (other variants have been studied in, e.g., [13]). Assuming an infinite set of *names* $(x, y, z, \dots)$, the set of *processes* $(P, Q, \dots)$ in CP and HCP is defined as follows:

$$P, Q ::= \mathbf{0} \mid (\nu xy)P \mid P \mid Q \mid [x \leftrightarrow y] \mid x[y].P \mid x(y).P \mid x[\,].P \mid x().P$$

Most constructs are similar to those in SP; differences are the forwarder process $[x \leftrightarrow y]$, which equates x and y, and process $x[y].P$, which denotes the output of the private name y along x, with continuation P. Also, the empty output is now denoted $x[\,].P$. In $x[y].P$ and $x(y).P$, the name y is bound in P. We write $P\{x/y\}$ to denote the capture-avoiding substitution of y for x in P. We write $\mathsf{fn}(P)$ to denote the free names of P. We use π to range over prefixes: $x[y], x(y), x(), x[\,]$.

The types assigned to names correspond to formulas of classical linear logic (CLL):

$$A, B ::= \mathbf{1} \mid \bot \mid A \otimes B \mid A \parr B$$

The assignment $x : A$ says that x follows the input/output interactions described by A. Assignments $x : A \otimes B$ and $x : A \parr B$ are read as sending and receiving an object of type A along x, with continuation B, respectively. There is a duality in the interpretation of the following pairs: $\otimes$ and $\parr$; $\mathbf{1}$ and $\bot$. Formally, the dual type of A, denoted $A^{\perp}$, is defined as

$$\mathbf{1}^{\perp} \triangleq \bot \qquad \bot^{\perp} \triangleq \mathbf{1} \qquad (A \otimes B)^{\perp} \triangleq A^{\perp} \parr B^{\perp} \qquad (A \parr B)^{\perp} \triangleq A^{\perp} \otimes B^{\perp}$$

Let Γ, Δ range over *environments*, unordered collections of assignments. Given an environment $\Delta = x_1 : A_1, \dots, x_n : A_n$, its domain, written $dom(\Delta)$, is the set $\{x_1, \dots, x_n\}$.

▶ **Definition 11** (CP). *Typing judgements for* CP *are of the form $P \vdash_{\mathtt{C}} \Gamma$, with typing rules as given in Figure 4. We shall write $P \in$ CP if $P \vdash_{\mathtt{C}} \Gamma$, for some Γ.*

Observe how typing in CP induces a specific shape for output processes: $x[y].(P \mid Q)$. This is not the case in HCP, where processes are described by *hyperenvironments*, unordered collections of environments: $\mathcal{H}, \mathcal{G} ::= \Gamma_1 \parallel \cdots \parallel \Gamma_n$.

▶ **Definition 12** (HCP). *Typing judgements for* HCP *are of the form $P \vdash_{\mathtt{H}} \mathcal{G}$, with typing rules as given in Figure 5. We shall write $P \in$ HCP if $P \vdash_{\mathtt{H}} \mathcal{G}$, for some $\mathcal{G}$.*

CP and HCP differ in process composition, which is key to exclude the circular dependencies that lead to deadlocks: it is realized by Rule C-Cut in CP and Rule H-Cut in HCP. The former requires two premises describing processes that share exactly one session; typing

$$\textsc{H-Mix}_2\ \frac{P \vdash_{\mathtt{H}} \mathcal{H} \quad Q \vdash_{\mathtt{H}} \mathcal{G}}{P \mid Q \vdash_{\mathtt{H}} \mathcal{H} \parallel \mathcal{G}} \qquad \textsc{H-Mix}_0\ \frac{}{\mathbf{0} \vdash_{\mathtt{H}} \cdot} \qquad \textsc{H-id}\ \frac{}{[x \leftrightarrow y] \vdash_{\mathtt{H}} x : A, y : A^{\perp}} \qquad \text{H-}\bot\ \frac{P \vdash_{\mathtt{H}} \mathcal{H} \parallel \Delta}{x().P \vdash_{\mathtt{H}} \mathcal{H} \parallel \Delta, x : \bot} \qquad \text{H-}\mathbf{1}\ \frac{P \vdash_{\mathtt{H}} \mathcal{H}}{x[].P \vdash_{\mathtt{H}} \mathcal{H} \parallel x : \mathbf{1}}$$

$$\text{H-}\otimes\ \frac{P \vdash_{\mathtt{H}} \mathcal{H} \parallel \Delta, y : A \parallel \Gamma, x : B}{x[y].P \vdash_{\mathtt{H}} \mathcal{H} \parallel \Delta, \Gamma, x : A \otimes B} \qquad \text{H-}\parr\ \frac{P \vdash_{\mathtt{H}} \mathcal{G} \parallel \Delta, y : B, x : A}{x(y).P \vdash_{\mathtt{H}} \mathcal{G} \parallel \Delta, x : A \parr B} \qquad \textsc{H-Cut}\ \frac{P \vdash_{\mathtt{H}} \mathcal{H} \parallel \Delta, x : A \parallel \Gamma, y : A^{\perp}}{(\nu xy)P \vdash_{\mathtt{H}} \mathcal{H} \parallel \Delta, \Gamma}$$

Figure 5 HCP: Typing rules for processes.

involves both composition and restriction and induces tree-like process topologies. The latter involves one premise only, which describes a process in which the separation between x and y is made explicit by "$\parallel$" at the level of hyperenvironments; typing involves restriction only.

Every P typable in CP is also typable in HCP, but the converse does not hold. We have:

▶ **Lemma 13** (Relation Between CP and HCP [23]). *If $P \vdash_{\mathtt{C}} \Gamma$ then $P \vdash_{\mathtt{H}} \Gamma$.*

▶ **Example 14** (The converse of Lemma 13 does not hold). To illustrate a process typable in HCP but not in CP, consider $P_{14} \triangleq (\nu x_1 y_1)(\nu x_2 y_2)(\nu v_1 k_1)(\nu v_2 k_2)(P'_{14} \mid P''_{14})$, where:

$$P'_{14} \triangleq x_1(z_1).x_2(z_2).x_1().x_2().z_1().z_2().\mathbf{0}$$

$$P''_{14} \triangleq y_1[v'_1].([v'_1 \leftrightarrow v_1] \mid y_2[v'_2].([v'_2 \leftrightarrow v_2] \mid C)) \qquad C \triangleq y_1[].\mathbf{0} \mid y_2[].\mathbf{0} \mid k_1[].\mathbf{0} \mid k_2[].\mathbf{0}$$

We have $P_{14} \vdash_{\mathtt{H}} \cdot$. First, $P'_{14} \vdash_{\mathtt{H}} x_1 : \bot \parr \bot, x_2 : \bot \parr \bot$ follows from several applications of Rules H-$\bot$ and H-$\parr$. To type P''_{14}, we first obtain $C \vdash_{\mathtt{H}} y_1 : \mathbf{1} \parallel y_2 : \mathbf{1} \parallel k_1 : \mathbf{1} \parallel k_2 : \mathbf{1}$, crucially using Rule H-Mix$_2$ several times to compose the four independent processes. Because $[v'_2 \leftrightarrow v_2] \vdash_{\mathtt{H}} v'_2 : \mathbf{1}, v_2 : \bot$, by Rules H-Mix$_2$ and H-$\otimes$ we have: $y_2[v'_2].([v'_2 \leftrightarrow v_2] \mid C) \vdash_{\mathtt{H}} y_1 : \mathbf{1} \parallel y_2 : \mathbf{1} \otimes \mathbf{1}, v_2 : \bot \parallel k_1 : \mathbf{1} \parallel k_2 : \mathbf{1}$. We proceed similarly to type the output on y_1 and complete the typing of P''_{14}. Finally, to compose P'_{14} and P''_{14} we apply H-Mix$_2$ and H-Cut.

Now, C cannot be typed in CP: there is no rule similar to Rule H-Mix$_2$, which can compose independent processes. Hence, P_{14} is not typable in CP.

In general, the composition of two processes that share more than one session cannot be typed in CP. Now consider Q_{14}:

$$\begin{aligned} Q_{14} \triangleq (\nu v_2 k_2)((\nu v_1 k_1)((\nu x_2 y_2)((\nu x_1 y_1)(P'_{14} \mid y_1[v'_1].([v'_1 \leftrightarrow v_1] \mid y_1[].\mathbf{0}) \\ \mid y_2[v'_2].([v'_2 \leftrightarrow v_2] \mid y_2[].\mathbf{0})) \mid k_1[].\mathbf{0}) \mid k_2[].\mathbf{0}) \end{aligned}$$

Observe how Q_{14} modifies P_{14} by keeping P'_{14} unchanged and dividing the two outputs in P''_{14} into threads that can be composed separately using Rule C-Cut. We have $Q_{14} \vdash_{\mathtt{C}} \cdot$.

Given a $P \vdash_{\mathtt{H}} \mathcal{G}$, with $\mathcal{G} = \Gamma_1 \parallel \cdots \parallel \Gamma_n$, we are interested in partitioning the free names in P according to the environments $\Gamma_1, \ldots, \Gamma_n$. These *name partitions* describe which names are used by each parallel component of P; they are defined considering all the different hyperenvironments under which P is typable.

▶ **Definition 15** (HCP: Name Partitions). *Given $\mathcal{G} = \Gamma_1 \parallel \cdots \parallel \Gamma_n$, we define* $\mathsf{Part}(\mathcal{G}) = \{dom(\Gamma_1), \ldots, dom(\Gamma_n)\}$. *Given P, its* name partitions *are* $\mathsf{np}(P) = \{\mathsf{Part}(\mathcal{G}) \mid P \vdash_{\mathtt{H}} \mathcal{G}\}$.

$x *_P y$ holds whenever there is a $G \in \mathsf{np}(P)$ such that x and y belong to different proper subsets of G. Also, $x \circledast_P y$ holds whenever x and y belong to the same proper subset of G.

▶ **Example 16.** Consider the process $P_{16} \triangleq x().(z[].\mathbf{0} \mid y[].\mathbf{0})$: it can be typed under the hyperenvironments $\mathcal{G} \triangleq x : \bot, z : \mathbf{1} \parallel y : \mathbf{1}$ and $\mathcal{H} \triangleq z : \mathbf{1} \parallel x : \bot, y : \mathbf{1}$. Given this, $\mathsf{np}(P_{16}) = \{\{\{x, z\}, \{y\}\}, \{\{z\}, \{x, y\}\}\}$, and we have, e.g., $y *_{P_{16}} z$ and $x \circledast_{P_{16}} y$.

$$\frac{}{[x \leftrightarrow y] \xrightarrow{[x\leftrightarrow y]} \mathbf{0}}\,[x \leftrightarrow y] \qquad \frac{}{[x \leftrightarrow y] \xrightarrow{[y\leftrightarrow x]} \mathbf{0}}\,[x \leftrightarrow y] \qquad \frac{\pi \in \{x[y], x(y), x(), x[\,]\}}{\pi.P \xrightarrow{\pi} P}\,\pi$$

$$\frac{P \xrightarrow{l} P' \quad \mathsf{bn}(l) \cap \mathsf{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{l} P' \mid Q}\,\text{PAR}_1 \qquad \frac{Q \xrightarrow{l} Q' \quad \mathsf{bn}(l) \cap \mathsf{fn}(P) = \emptyset}{P \mid Q \xrightarrow{l} P \mid Q'}\,\text{PAR}_2 \qquad \frac{P \xrightarrow{[y\leftrightarrow z]} P'}{(\nu\, xy)P \xrightarrow{\tau} P'\{x/z\}}\,\textsc{AxCut}$$

$$\frac{P \xrightarrow{x[\,] \mid y()} P'}{(\nu\, xy)P \xrightarrow{\tau} P'}\,\bot\mathbf{1} \qquad \frac{P =_\alpha Q \quad Q \xrightarrow{l} R}{P \xrightarrow{l} R}\,=_\alpha \qquad \frac{P \xrightarrow{l} P' \quad x, y \notin \mathsf{cn}(l) \quad x *_{P'} y}{(\nu\, xy)P \xrightarrow{l} (\nu\, xy)P'}\,\textsc{Res}$$

$$\frac{P \xrightarrow{x[x'] \mid y(y')} P'}{(\nu\, xy)P \xrightarrow{\tau} (\nu\, xy)(\nu\, x'y')P'}\,\otimes\parr \qquad \frac{P \xrightarrow{l} P' \quad Q \xrightarrow{l'} Q' \quad \mathsf{bn}(l) \cap \mathsf{bn}(l') = \emptyset}{P \mid Q \xrightarrow{l \mid l'} P' \mid Q'}\,\text{SYN}$$

$$\frac{P \xrightarrow{l} P' \quad x \notin \mathsf{cn}(l)}{x[\,].P \xrightarrow{l} x[\,].P'}\,\pi_1 \qquad \frac{P \xrightarrow{l} P' \quad \mathsf{fn}(P') \neq \emptyset}{x().P \xrightarrow{l} x().P'}\,\pi_2 \qquad \frac{P \xrightarrow{l} P' \quad x, x' \notin \mathsf{cn}(l) \quad x *_{P'} x'}{x[x'].P \xrightarrow{l} x[x'].P'}\,\otimes$$

$$\frac{P \xrightarrow{l} P' \quad x, x' \notin \mathsf{cn}(l) \quad x \circledast_{P'} x'}{x(x').P \xrightarrow{l} x(x').P'}\,\parr \qquad \frac{\pi.P \xrightarrow{l} \pi.P' \quad \mathsf{fn}(\pi) *_{\pi.P} \mathsf{fn}(l)}{\pi.P \xrightarrow{\pi \mid l} P'}\,\mid \pi$$

Figure 6 The LTS for HCP (Definition 17) includes rules for actions and communication (top) and for delayed actions and self-synchronizations (bottom).

The semantics of HCP is given in terms of a Labeled Transition System (LTS):

▶ **Definition 17** (HCP: LTS). *The binary relation on* HCP *processes* $\xrightarrow{l}$ *is defined by the rules in Figure 6, with action labels defined as follows:*

$$l, l' ::= [x \leftrightarrow y] \mid x[y] \mid x(y) \mid x() \mid x[\,] \mid l \mid l \mid \tau$$

In the following, $\mathsf{fn}(l)$*,* $\mathsf{bn}(l)$*, and* $\mathsf{cn}(l)$ *denote the free, bound, and channel names of* l*, respectively. They are defined as follows:* $\mathsf{fn}([x \leftrightarrow y]) \triangleq \{x, y\}$*,* $\mathsf{fn}(\tau) = \mathsf{bn}(\tau) \triangleq \emptyset$*,* $\mathsf{fn}(x[y]) = \mathsf{fn}(x(y)) = \mathsf{fn}(x()) = \mathsf{fn}(x()) \triangleq x$*, and* $\mathsf{bn}(x[y]) = \mathsf{bn}(x(y)) \triangleq \{y\}$*. Also,* $\mathsf{fn}(l_1 \mid l_2) \triangleq \mathsf{fn}(l_1) \cup \mathsf{fn}(l_2)$*, and* $\mathsf{bn}(l_1 \mid l_2) \triangleq \mathsf{bn}(l_1) \cup \mathsf{bn}(l_2)$*. Moreover,* $\mathsf{cn}(l) \triangleq \mathsf{fn}(l) \cup \mathsf{bn}(l)$*.*

The rules in Figure 6 are inspired by proof-theoretical transformations on HCP's proof derivations [23]; they can be divided into two classes. The rules above the line are usual formulations for actions, structural rules, and communication. Actions can appear in parallel in a label, and compatible actions in parallel induce a τ-transition. The five rules below the line formalize *delayed actions* and *self-synchronizations*. Intuitively, the first four rules allow to observe an action l in a delayed style; such an action must be independent from the prefix at the top-level (on x). The last rule, $\mid \pi$, implements self-synchronization for a process $\pi.P$: the (top-level) prefix π is considered in parallel with a (delayed) action l emanating from P.

▶ **Example 18.** To illustrate the LTS of HCP, recall the processes P_1 and P_2 from § 1. In HCP, they are denoted as: $P_{18} \triangleq (\nu\, xx')(x[\,].x'().\mathbf{0})$ and $Q_{18} \triangleq (\nu\, xx')(\nu\, yy')(y[\,].x().\mathbf{0} \mid x'[\,].y'().\mathbf{0})$. Consider the following type derivation, parametric on w, z:

$$\Pi_{w,z} = \cfrac{\cfrac{\mathbf{0} \vdash_{\mathtt{H}}}{w().\mathbf{0} \vdash_{\mathtt{H}} w : \bot}\,\text{H-}\bot}{z[\,].w().\mathbf{0} \vdash_{\mathtt{H}} w : \bot \parallel z : \mathbf{1}}\,\text{H-}\mathbf{1}$$

Thus, P_{18} is typed as $\Pi_{x',x}$ followed by an application of Rule H-Cut. Also, Q_{18} is typed by $\Pi_{x,y}$ and $\Pi_{y',x'}$ followed by an application of Rules H-Mix$_2$ and H-Cut. Because Rule C-1 only types empty outputs followed by $\mathbf{0}$, neither P_{18} nor Q_{18} are typable in CP.

By Rules $\mid \pi$ and $\bot\mathbf{1}$ in Figure 6, $P_{18} \xrightarrow{\tau} \mathbf{0}$. Similarly, $y[\,].x().\mathbf{0} \mid x'[\,].y'().\mathbf{0} \xrightarrow{x() \mid x'[\,]} y[\,].\mathbf{0} \mid y'().\mathbf{0}$ by Rules π, π_1 and SYN. Finally, by Rule $\bot\mathbf{1}$, $Q_{18} \xrightarrow{\tau} (\nu\, yy')(y[\,].\mathbf{0} \mid y'().\mathbf{0})$.

We now recall some useful notions and results from [23].

▶ **Definition 19** (HCP: Weak LTS and Bisimilarity [23]). *$\overset{l}{\Rightarrow}$ is the smallest relation such that: (1) $P \overset{\tau}{\Rightarrow} P$ for all P; (2) and if $P \overset{\tau}{\Rightarrow} P'$, $P' \xrightarrow{l} Q'$, and $Q' \overset{\tau}{\Rightarrow} Q$, then $P \overset{l}{\Rightarrow} Q$.*

A symmetric relation $\mathcal{R}$ is a bisimulation if $P \,\mathcal{R}\, Q$ implies that if $P \overset{l}{\Rightarrow} P'$ then $Q \overset{l}{\Rightarrow} Q'$ for some Q' such that $P' \,\mathcal{R}\, Q'$. Bisimilarity is the largest relation $\approx_{\mathtt{H}}$ that is a bisimulation.

Well-typed processes satisfy readiness, defined as follows:

▶ **Theorem 20** (HCP: Readiness [23]). *Let $P \vdash_{\mathtt{H}} \Gamma_1 \parallel \cdots \parallel \Gamma_n$. For every $i \in [1,n]$, there exist $x \in dom(\Gamma_i)$, l_x, and P' such that $P \overset{l_x}{\Longrightarrow} P'$.*

In HCP, transitions associated to sessions in different partitions can be fired in parallel:

▶ **Lemma 21** ([23]). *If $P \vdash_{\mathtt{H}} \mathcal{G} \parallel \Gamma, x : A \parallel \Delta, y : B$, $P \overset{l_x}{\Longrightarrow} P'$, and $P \overset{l_y}{\Longrightarrow} P''$, then there exists Q s.t. $P \overset{l_x \mid l_y}{\Longrightarrow} Q$ (up to α-renaming).*

▶ **Definition 22** (HCP: Structural Congruence). *Let $\equiv_{\mathtt{H}}$ be the smallest congruence relation generated by the axioms:*

$$\begin{array}{rclcrcl}
P \mid Q & \equiv_{\mathtt{H}} & Q \mid P & & P \mid \mathbf{0} & \equiv_{\mathtt{H}} & P \\
P \mid (Q \mid R) & \equiv_{\mathtt{H}} & (P \mid Q) \mid R & & (\nu\, xy)P & \equiv_{\mathtt{H}} & (\nu\, yx)P \\
(\nu\, xy)P \mid Q & \equiv_{\mathtt{H}} & (\nu\, xy)(P \mid Q) \quad \text{if } x, y \notin \mathsf{fn}(Q) & & (\nu\, xy)(\nu\, wz)P & \equiv_{\mathtt{H}} & (\nu\, wz)(\nu\, xy)P
\end{array}$$

Because of Readiness (Theorem 20), HCP also satisfies *progress*:

▶ **Corollary 23** (HCP: Progress [23]). *If $P \in$ HCP and $P \not\equiv_{\mathtt{H}} \mathbf{0}$, then $P \xrightarrow{l} P'$ for some l, P'.*

In order to relate with reduction-based semantics (in SP), we define the following:

▶ **Definition 24** (HCP: Reduction). *We write $\rightarrow^*$ for the reflexive, transitive closure of $\xrightarrow{\tau}$.*

▶ **Lemma 25.** *Let $P \in$ HCP s.t. $P \vdash_{\mathtt{H}} \cdot$, then exists P' s.t. $P \rightarrow^* P' \equiv_{\mathtt{H}} \mathbf{0}$.*

Proof sketch. We prove it in two steps: (1) We prove that if $P \not\equiv_{\mathtt{H}} \mathbf{0}$, then there exists P' s.t. $P \xrightarrow{\tau} P'$, and $P' \vdash_{\mathtt{H}} \cdot$. This ensures the existence of a τ-transition in every step. (2) We then prove that $\mathbf{0}$ can be reached in finitely many steps. See [19] for details. ◀

Inspired by Lemma 25 we have the following definition of deadlock freedom for HCP.

▶ **Definition 26** (HCP: Deadlock Freedom). *Let $P \in$ HCP. We say that P is deadlock free, written $\mathcal{D}_{\mathbf{H}}(P)$, iff there exists P' s.t. $P \rightarrow^* P' \equiv_{\mathtt{H}} \mathbf{0}$.*

HCP has a *denotational semantics* that interprets well-typed processes as sets of possible interactions on their free names. Given a process P, its set of denotations is written $\|P\|$. Denotational equivalence, noted $\doteq$, is the typed relation defined as: $P \doteq Q$ iff $\|P\| = \|Q\|$. We shall rely on denotational equivalence; its exact formulation, given in [19], is interesting but not crucial for following our results. We notice that $\equiv_{\mathtt{H}}$ is sound wrt $\doteq$ and that our results are reflected by bisimilarity, as ensured by the following result:

▶ **Theorem 27** (HCP: Full Abstraction [23]). *For well-typed processes: $\approx_{\mathtt{H}} = \doteq$.*

3 The Role of Delayed Actions and Self-Synchronizations in DF

We have seen that SP defines a flexible typing discipline that admits deadlocked and deadlock-free processes. Also, we saw the way in which CP and HCP enforce DF by typing, and that the latter's LTS allows for delayed actions and self-synchronizations. By defining two classes of SP processes, $\mathbb{C}$ and $\mathbb{H}$, which capture typability as in CP and HCP, respectively (Definition 34), we study whether delayed actions and self-synchronizations influence DF.

We proceed as follows. As we have seen, $\mathcal{D}_{\mathbf{H}}(P)$ means that P is deadlock free by considering all sources of behavior, including delayed actions and self-synchronizations; we shall also define $\mathcal{D}^{\mathsf{R}}_{\mathbf{H}}(P)$ to mean that P is deadlock free by considering *regular transitions only*. Our main result is Corollary 53, which ensures: (1) delayed actions and self-synchronizations are *inessential* when it comes to DF, and (2) given an encoding $[\![\cdot]\!] : \mathsf{SP} \to \mathsf{HCP}$, for a process $P \in \mathsf{SP}$ such that $\mathcal{D}_{\mathbf{H}}([\![P]\!])$, there is a P' that *simulates* P, and $\mathcal{D}^{\mathsf{R}}_{\mathbf{H}}([\![P']\!])$.

3.1 From SP to HCP

We start by translating processes in SP into processes in HCP:

▶ **Definition 28.** *Let $P \in \mathsf{SP}$. The encoding $[\![\cdot]\!] : \mathsf{SP} \to \mathsf{HCP}$ is a homomorphism except for the case of free output, defined as $[\![\overline{x}\langle v\rangle.P]\!] \triangleq x[y].([y \leftrightarrow v] \mid [\![P]\!])$. For types, we define:*

$$[\![\mathsf{end}_?]\!] \triangleq \bot \qquad [\![\mathsf{end}_!]\!] \triangleq \mathbf{1} \qquad [\![?T.S]\!] \triangleq [\![T]\!] \parr [\![S]\!] \qquad [\![!T.S]\!] \triangleq [\![T]\!]^{\perp} \otimes [\![S]\!]$$

For typing environments, we define: $[\![\emptyset]\!] \triangleq \emptyset$ and $[\![\Gamma, x : T]\!] \triangleq [\![\Gamma]\!], x : [\![T]\!]$.

Regular Transitions do not depend on self-synchronizations or delayed actions:

▶ **Definition 29** (HCP: Regular Transitions). *A derivation T for $P \xrightarrow{l} Q$ is denoted $T : P \xrightarrow{l} Q$. We define $\mathsf{RT}(T)$ iff T is derived only using rules above the horizontal line in Figure 6.*

▶ **Example 30.** Recall the processes P_{18} and Q_{18} from Example 18, which use self-synchronizations and delayed actions, respectively: we have that $T_1 : P_{18} \xrightarrow{\tau} \mathbf{0}$, and $T_2 : Q_{18} \xrightarrow{\tau} (\nu\, yy')(y[\,].\mathbf{0} \mid y'().\mathbf{0})$. Therefore, $\neg\mathsf{RT}(T_1)$ and $\neg\mathsf{RT}(T_2)$.

The encoding of free output induces transitions involving the forwarder process, not present in the source SP process. We handle the substitutions induced by forwarders in Rule AxCut (Figure 6) as follows:

▶ **Definition 31** (HCP: Ready-for-Substitution). *Let $P \in \mathsf{HCP}$. We say that $P = (\nu\, xy)P'$ is ready for substitution of z for x, written $\mathsf{RS}_{z,x}(P)$, if there exists $z \in \mathsf{fn}(P)$ and P' s.t. $P \xrightarrow{[y \leftrightarrow z]} P'$, and $(\nu\, xy)P \xrightarrow{\tau} P'\{x/z\}$. Otherwise, we write $\neg\mathsf{RS}(P)$.*

We may now define transitions for processes that are ready for a substitution.

▶ **Definition 32** (HCP: Eager-to-Rename). *The relation $\xrightarrow[\mathsf{ER}]{l}$ on HCP processes is defined as:*

$$(P \xrightarrow{l} Q) \wedge \mathsf{RS}_{x,z}(Q) \Rightarrow P \xrightarrow[\mathsf{ER}]{l} Q\{x/z\} \qquad \textit{and} \qquad (P \xrightarrow{l} Q) \wedge \neg\mathsf{RS}(Q) \Rightarrow P \xrightarrow[\mathsf{ER}]{l} Q$$

Intuitively, $\xrightarrow[\mathsf{ER}]{l}$ executes outputs and forwarders in one single step. When $[\![P]\!]$ only relies on regular transitions we have the following operational correspondence result.

▶ **Theorem 33** ($[\![\cdot]\!]$: Operational Correspondence)**.** *Let $P \in \mathbb{H}$.*

1. *If $T : [\![P]\!] \xrightarrow[\mathtt{ER}]{\tau} R$ and $\mathsf{RT}(T)$, then $P \longrightarrow Q$ s.t. $[\![Q]\!] \equiv_{\mathtt{H}} R$.*
2. *If $P \longrightarrow Q$, then $[\![P]\!] \xrightarrow[\mathtt{ER}]{\tau} R$ s.t. $R \equiv_{\mathtt{H}} [\![Q]\!]$.*

Proof sketch. Part (1) is proven by case analysis in $\xrightarrow{\tau}$, and (2) is proven by induction in the derivation of $\longrightarrow$. See [19] for details. ◀

Following Dardha and Pérez [9], we now define $\mathbb{C}$ and $\mathbb{H}$ as the classes of SP processes that are also typable in CP and HCP, respectively:

▶ **Definition 34** (Classes $\mathbb{C}$ and $\mathbb{H}$)**.** *The classes $\mathbb{C}$ and $\mathbb{H}$ of SP processes are defined as:*

$$\mathbb{C} \triangleq \{P \in \mathsf{SP} \mid \Gamma \vdash_{\mathtt{S}} P \wedge [\![P]\!] \vdash_{\mathtt{C}} [\![\Gamma]\!]\}$$
$$\mathbb{H} \triangleq \{P \in \mathsf{SP} \mid \Gamma \vdash_{\mathtt{S}} P \wedge \Gamma = \Gamma_1, \ldots, \Gamma_n \wedge [\![P]\!] \vdash_{\mathtt{H}} [\![\Gamma_1]\!] \parallel \cdots \parallel [\![\Gamma_n]\!]\}$$

Hence, in writing "$P \in \mathbb{H}$" we are not referring to an HCP process directly, but rather to a SP process P (in the sense of § 2.1) that is related (up to $[\![\cdot]\!]$) to some (typable) HCP process. This is how SP offers a unifying framework for comparing different type systems. As a direct consequence of Lemma 13 and Example 14 we have that $\mathbb{C} \subsetneq \mathbb{H}$.

3.2 A Refined Account of $\mathbb{H}$, Based on Deadlock Freedom

As explained in the previous section, HCP enjoys liveness properties corresponding to the LTS. If we focus only on τ-transitions (cf. Definition 24) we obtain the following results.

We extend Definition 29 (regular transitions) to sequences of transitions, as follows:

▶ **Definition 35** (HCP: Regular Sequence of Transitions)**.** *Assume $P_1, \ldots, P_n$ in HCP and transitions $T_i : P_i \xrightarrow{l_i} P_{i+1}$. We say that a transition sequence $\rho : P \xrightarrow{l_1} \cdots \xrightarrow{l_n} P_n$ transitions regularly iff $\mathsf{RT}(T_i)$ for all $i = 1, \ldots, n$. We write $\mathsf{RT}(\rho)$.*

We are interested in processes whose transitions are regular, thus we refine deadlock freedom in HCP (Definition 26) as follows:

▶ **Definition 36** (HCP: Deadlock Freedom based on Regular Transitions)**.** *Let $P \in$ HCP. We say that P is deadlock free without delayed actions and self-synchronizations, written $\mathcal{D}^{\mathsf{R}}_{\mathbf{H}}(P)$, iff $\mathcal{D}_{\mathbf{H}}(P)$, $\rho : P \rightarrow^* P' \equiv_{\mathtt{H}} \mathbf{0}$, and $\mathsf{RT}(\rho)$.*

In turn, Definitions 26 and 36 allow us to refine the definition of $\mathbb{H}$ (Definition 34):

▶ **Definition 37** (HCP: Sub-classes of $\mathbb{H}$)**.** *We define the following classes of DF processes:*

$$\mathbb{H}_{\mathsf{F}} \triangleq \{P \in \mathbb{H} \mid \mathcal{D}_{\mathbf{H}}([\![P]\!])\} \qquad \mathbb{H}_{\mathsf{R}} \triangleq \{P \in \mathbb{H} \mid \mathcal{D}^{\mathsf{R}}_{\mathbf{H}}([\![P]\!])\}$$

By Definitions 34 and 37 we have $\mathbb{H}_{\mathsf{R}} \subsetneq \mathbb{H}_{\mathsf{F}} \subsetneq \mathbb{H}$.

▶ **Example 38.** Consider the SP processes:

$$P_{38} \triangleq (\nu\, wz)(\overline{w}\langle\rangle.(\nu\, xy)(\overline{x}\langle\rangle.\mathbf{0} \mid y().z().\mathbf{0})) \quad Q_{38} \triangleq (\nu\, wz)(\overline{w}\langle\rangle.\mathbf{0} \mid (\nu\, xy)(\overline{x}\langle\rangle.\mathbf{0} \mid z().y().\mathbf{0}))$$

We have $P_{38} \in \mathbb{H}_{\mathsf{F}} \setminus \mathbb{H}_{\mathsf{R}}$ and $Q_{38} \in \mathbb{H}_{\mathsf{R}}$. Although $\neg\mathcal{D}_{\mathbf{S}}(P_{38})$, the HCP process $[\![P_{38}]\!]$ has a τ-transition due to a delayed action, and therefore $\mathcal{D}_{\mathbf{H}}([\![P_{38}]\!])$.

▶ **Theorem 39.** *If $P \in \mathbb{H}_{\mathsf{R}}$, then $\mathcal{D}_{\mathbf{S}}(P)$.*

Proof. It follows from Theorem 33 and Definitions 9 and 37. ◀

$$\begin{array}{cc}
x().x'().P \simeq_{cc} x'().x().P & x().x'(y').P \simeq_{cc} x'(y').x().P \\
x(y).x'(y').P \simeq_{cc} x'(y').x(y).P & x().x'[y'].P \simeq_{cc} x'[y'].x().P \\
x[y].x'(y').P \simeq_{cc} x'(y').x[y].P & x[\,].x'(y').P \simeq_{cc} x'(y').x[\,].P \\
x[y].x'[y'].P \simeq_{cc} x'[y'].x[y].P & x[\,].x'[y'].P \simeq_{cc} x'[y'].x[\,].P \\
x[\,].x'[\,].P \simeq_{cc} x'[\,].x[\,].P & x[\,].x'().P \simeq_{cc} x'().x[\,].P \\
(\nu\, xy)(x'().P) \simeq_{cc} x'().(\nu\, xy)(P) & (\nu\, xy)(x'[\,].P) \simeq_{cc} x'[\,].(\nu\, xy)(P) \\
(\nu\, xy)(x'(y').P) \simeq_{cc} x'(y').(\nu\, xy)(P) & (\nu\, xy)(x'[y'].P) \simeq_{cc} x'[y'].(\nu\, xy)(P)
\end{array}$$

Figure 7 HCP: Commuting Conversions (Definition 40).

From $\mathbb{H}_\mathsf{F}$ to $\mathbb{H}_\mathsf{R}$ via process optimizations. We now embark to show how to transform processes in $\mathbb{H}_\mathsf{F}$ (where self-synchronizations and delayed actions are used) into processes in $\mathbb{H}_\mathsf{R}$ (where they are not). Intuitively, we would like to transform, e.g.,

$$(\nu\, wz)(\overline{w}\langle\,\rangle.(\nu\, xy)(\overline{x}\langle\,\rangle.\mathbf{0} \mid y().z().\mathbf{0})) \quad \text{into} \quad (\nu\, wz)(\overline{w}\langle\,\rangle.\mathbf{0} \mid (\nu\, xy)(\overline{x}\langle\,\rangle.\mathbf{0} \mid z().y().\mathbf{0}))$$

The transformation from $\mathbb{H}_\mathsf{F}$ to $\mathbb{H}_\mathsf{R}$ is formalized by Theorem 47. To achieve it, we leverage two existing ingredients. The *first ingredient* is the notion of *commuting conversions*: sound transformations on processes that involve actions from independent sessions (cf. [10, 30]).

▶ **Definition 40** (HCP: Commuting Conversions). *We define $\simeq_{cc}$ as the smallest congruence on* HCP *processes induced by the equalities in Figure 7.*

The transformations induced by $\simeq_{cc}$ have a proof-theoretical origin, and so they are type-preserving, which ensures that they do not alter the causality relations on processes. We can show that that $\simeq_{cc}$ is sound wrt $\approx_\mathtt{H}$, i.e., if $P \simeq_{cc} Q$ then $P \approx_\mathtt{H} Q$.

▶ **Example 41.** Recall processes P'_{14}, P''_{14}, and P_{14} from Example 14. We illustrate the optimizations on processes induced by commuting conversions by combining P''_{14} and the process $Q_{41} \triangleq x_2(z_2).x_1(z_1).x_1().x_2().z_1().z_2().\mathbf{0}$, as follows:

$$P_{41} \triangleq (\nu\, x_1y_1)(\nu\, x_2y_2)(\nu\, v_1k_1)(\nu\, v_2k_2)(Q_{41} \mid P''_{14})$$

Note that a τ-transition along the sessions x_1y_1 or x_2y_2 in P_{41} would depend on a delayed action. Now, we can use a commuting conversion to swap the order of the inputs, effectively anticipating the input on x_1: $Q_{41} \simeq_{cc} x_1(z_1).x_2(z_2).x_1().x_2().z_1().z_2().\mathbf{0} = P'_{14}$; therefore, $P_{41} \simeq_{cc} P_{14}$. Clearly, there is a transition sequence $\rho : P_{14} \to^* \equiv_\mathtt{H} \mathbf{0}$ s.t. $\mathsf{RT}(\rho)$.

The *second ingredient* is *disentanglement*, which also induces optimizations on processes:

▶ **Definition 42** (HCP: Disentanglement [22]). *Disentanglement is described by the smallest relation $\rightsquigarrow$ on (typed) processes closed under the rules in Figure 8. We write $\rightsquigarrow^*$ for the reflexive and transitive closure of $\rightsquigarrow$.*

Each of the process transformations in Figure 8 is justified by an operation on derivations; this way, e.g., for the second rule we have:

$$\dfrac{\dfrac{P \vdash_\mathtt{H} \mathcal{G} \parallel \Gamma \qquad Q \vdash_\mathtt{H} \mathcal{H}}{P \mid Q \vdash_\mathtt{H} \mathcal{G} \parallel \mathcal{H} \parallel \Gamma}\ \text{H-Mix}_2}{x().(P \mid Q) \vdash_\mathtt{H} \mathcal{G} \parallel \mathcal{H} \parallel \Gamma, x : \bot}\ \text{H-}\bot \quad \rightsquigarrow \quad \dfrac{\dfrac{P \vdash_\mathtt{H} \mathcal{G} \parallel \Gamma}{x().P \vdash_\mathtt{H} \mathcal{G} \parallel \Gamma, x : \bot}\ \text{H-}\bot \qquad Q \vdash_\mathtt{H} \mathcal{H}}{x().P \mid Q \vdash_\mathtt{H} \mathcal{G} \parallel \mathcal{H} \parallel \Gamma, x : \bot}\ \text{H-Mix}_2$$

$$\begin{array}{lcl}
x[\,].P \vdash_{\mathtt{H}} \mathcal{G} \parallel x : \mathbf{1} & \rightsquigarrow & x[\,].\mathbf{0} \mid P \vdash_{\mathtt{H}} \mathcal{G} \parallel x : \mathbf{1} \\
x().(P \mid Q) \vdash_{\mathtt{H}} \mathcal{G} \parallel \mathcal{H} \parallel \Gamma, x : \bot & \rightsquigarrow & x().P \mid Q \vdash_{\mathtt{H}} \mathcal{G} \parallel \mathcal{H} \parallel \Gamma, x : \bot \\
(\nu xy)(P \mid Q) \vdash_{\mathtt{H}} \mathcal{G} \parallel \mathcal{H} \parallel \Gamma \parallel \Delta & \rightsquigarrow & (\nu\, xy)P \mid Q \vdash_{\mathtt{H}} \mathcal{G} \parallel \mathcal{H} \parallel \Gamma \parallel \Delta \\
x[y].(P \mid Q) \vdash_{\mathtt{H}} \mathcal{G} \parallel \mathcal{H} \parallel \Gamma, \Delta, x : A \otimes B & \rightsquigarrow & x[y].P \mid Q \vdash_{\mathtt{H}} \mathcal{G} \parallel \mathcal{H} \parallel \Gamma, \Delta, x : A \otimes B \\
x(y).(P \mid Q) \vdash_{\mathtt{H}} \mathcal{G} \parallel \mathcal{H} \parallel \Gamma, \Delta, x : A \parr B & \rightsquigarrow & x(y).P \mid Q \vdash_{\mathtt{H}} \mathcal{G} \parallel \mathcal{H} \parallel \Gamma, \Delta, x : A \otimes B
\end{array}$$

Figure 8 HCP: Disentanglement (Definition 42). See [19] for details.

This way, disentanglement proceeds by the repeated application of the rules from Figure 8 to the derivation of $P \vdash_{\mathtt{H}} \Gamma_1 \parallel \cdots \parallel \Gamma_n$, which aims at moving downwards the instances of Rule H-Mix$_2$ (cf. Figure 5). If H-Mix$_2$ gets stuck in either H-Cut or H-$\otimes$, then it becomes an instance of C-Cut or C-$\otimes$, respectively (cf. Figure 4).

Disentanglement increases the parallelism in processes. This way, e.g., for P_{14} and Q_{14} (Example 14) we have: $P_{14} \rightsquigarrow^* Q_{14}$. Also, note that Rule H-**1** types $x[\,].P \vdash_{\mathtt{H}} \Gamma \parallel x : \mathbf{1}$, induces a discrepancy: at the process level, the rule places the empty output in sequential composition with P, whereas at the level of typing $x : \mathbf{1}$ is placed in its own partition, enforcing independence from all names in $\mathsf{fn}(P)$. Disentanglement solves this discrepancy. Consider, e.g., process $P_{18} = (\nu\, xx')(x[\,].x'().\mathbf{0})$: we have $P_{18} \rightsquigarrow^* (\nu\, xx')(x[\,].\mathbf{0} \mid x'().\mathbf{0})$.

▶ **Lemma 43** (Disentanglement [23]). *If there exists a derivation d of $P \vdash_{\mathtt{H}} \Gamma_1 \parallel \cdots \parallel \Gamma_n$, then there exist derivations $d_1, \ldots, d_n$ of $P_1 \vdash_{\mathtt{C}} \Gamma_1, \ldots, P_n \vdash_{\mathtt{C}} \Gamma_n$ in* CP, *and*

$$P \vdash_{\mathtt{H}} \Gamma_1 \parallel \cdots \parallel \Gamma_n \quad \rightsquigarrow^* \quad \dfrac{P_1 \vdash_{\mathtt{C}} \Gamma_1 \ \cdots\ P_n \vdash_{\mathtt{C}} \Gamma_n}{P_1 \mid \cdots \mid P_n \vdash_{\mathtt{H}} \Gamma_1 \parallel \cdots \parallel \Gamma_n}\ \textit{H-Mix}_2$$

where we use the double line to indicate multiple uses of the same rule.

The following result ensures that process observations are invariant under disentanglement. Moreover, since instances of H-Cut are turned into instances of C-Cut, disentangled processes do not feature occurrences of restriction between sessions in sequential composition.

▶ **Lemma 44.** *Suppose $P \in$ HCP. If $P \rightsquigarrow^* P'$ then $P \approx_{\mathtt{H}} P'$.*

Proof. We prove that if $P \rightsquigarrow^* P'$ then $P \doteq P'$ by induction on the length k of $\rightsquigarrow^*$. For the base case ($k = 1$) we proceed by a case analysis in the last rule applied in the type derivation of $P \vdash_{\mathtt{H}} \mathcal{G}$. The fact that $P \approx_{\mathtt{H}} P'$ follows from Theorem 27. ◀

Given a process P, the rules for disentanglement can be applied in any order; we therefore define a set of disentangled processes:

▶ **Definition 45** (HCP: Disentangled Processes). *Given $P \in$ HCP, the set $\textsc{Dist}(P)$ is defined as follows:*

$$\textsc{Dist}(P) \triangleq \left\{ P' \in \text{HCP} \;\middle|\; \begin{array}{c} P \rightsquigarrow^* P', \\ P' = P_1 \mid \cdots \mid P_n \vdash_{\mathtt{H}} \Gamma_1 \parallel \cdots \parallel \Gamma_n, \quad P_1 \vdash_{\mathtt{C}} \Gamma_1, \ \cdots \ , P_n \vdash_{\mathtt{C}} \Gamma_n \end{array} \right\}$$

From Lemma 44 we have that all disentangled processes are observationally equivalent:

▶ **Lemma 46.** *Let $P \in$ HCP. If $Q, R \in \textsc{Dist}(P)$, then $Q \approx_{\mathtt{H}} R$.*

Proof. From Lemma 44, $P \approx_{\mathtt{H}} P'$ for all $P' \in \textsc{Dist}(P)$. By Definition 19, $\approx_{\mathtt{H}}$ is transitive. Hence, for all $Q, R \in \textsc{Dist}(P)$, $Q \approx_{\mathtt{H}} R$. ◀

We use disentanglement to prove that if $\mathcal{D}_{\mathbf{H}}(P)$ then there is a P' s.t. $\mathcal{D}^{\mathsf{R}}_{\mathbf{H}}(P')$, with $P \approx_{\mathtt{H}} P'$.

The following main result establishes that the notions of $\mathcal{D}_{\mathbf{H}}(\cdot)$ (Definition 26) and $\mathcal{D}^{\mathsf{R}}_{\mathbf{H}}(\cdot)$ (Definition 36) are equivalent, up to commuting conversions and disentanglement.

▶ **Theorem 47.** *For all processes $P \in \mathsf{HCP}$ s.t. $\mathcal{D}_{\mathbf{H}}(P)$, then there exist $P', Q \in \mathsf{HCP}$ s.t.: (1) $P' \in \textsc{Dist}(P)$, (2) $P' \simeq_{cc} Q$, (3) $P \approx_{\mathtt{H}} Q$, (4) $\mathcal{D}^{\mathsf{R}}_{\mathbf{H}}(Q)$.*

Proof sketch. Parts (1)-(3) follow from Definitions 40 and 45 and Lemma 44. For part (4) we show that self-synchronizations and delayed actions are eliminated by means of disentanglement and commuting conversions, respectively. See [19] for details. ◀

Theorem 47 concerns HCP processes. To lift this result to SP, we need some results.

▶ **Lemma 48.** *Let $P \in \mathbb{H}$. There exists $P' \in \mathbb{H}$ s.t.: (1) $[\![P]\!] \rightsquigarrow^* [\![P']\!]$. (2) $[\![P]\!] \simeq^*_{cc} [\![P']\!]$.*

Proof. The items follow by induction on the length of $\rightsquigarrow^*$ and $\simeq^*_{cc}$, respectively. For (1) in the base cases we proceed by a case analysis in the last rule applied in the type derivation of $[\![P]\!] \vdash_{\mathtt{H}} [\![\Gamma]\!]$; it shows the structure of $[\![P]\!]$ and by Definition 28 we obtain the structure of P. The proof finishes by applying a single step of $\rightsquigarrow$ to the type derivation of $[\![P]\!] \vdash_{\mathtt{H}} [\![\Gamma]\!]$ and replicating it to the derivation of $\Gamma \vdash_{\mathtt{S}} P$. The proof of (2) proceeds similarly. ◀

Lemma 48 ensures that the following definition is sound:

▶ **Definition 49** (Lifting Commuting Conversions and Disentanglement to SP)**.** *Let $P, P' \in \mathbb{H}$. We write $P \rightsquigarrow^* P'$ and $P \simeq^*_{cc} P'$, whenever $[\![P]\!] \rightsquigarrow^* [\![P']\!]$ and $[\![P]\!] \simeq^*_{cc} [\![P']\!]$ hold, respectively.*

Having disentanglement in SP, we aim to prove that the resulting disentangled process *simulates* the original process. To define simulation in SP, we need the following notion:

▶ **Definition 50** (SP: Labelled Reduction Semantics)**.** *Let $P \in \mathsf{SP}$, then $P \xRightarrow{\tau:xy} Q$ whenever:*
(1) $P \equiv (\nu\, xy)(\overline{x}\langle v\rangle.P' \mid y(z).Q' \mid R) \longrightarrow (\nu\, xy)(P' \mid Q'\{v/z\} \mid R) \equiv Q$, or (2) $P \equiv (\nu\, xy)(x().P' \mid \overline{y}\langle\rangle.Q' \mid R) \longrightarrow (\nu\, xy)(P' \mid Q' \mid R) \equiv Q$.

▶ **Definition 51** (SP: Simulation)**.** *A binary relation $\mathbb{S}$ in SP is a (typed) simulation if whenever $(P, Q) \in \mathbb{S}$ then $\Gamma \vdash_{\mathtt{S}} P$, $\Gamma \vdash_{\mathtt{S}} Q$ and $P \xRightarrow{\tau:xy} P'$ implies $Q \xRightarrow{\tau:xy} Q'$ for some Q' s.t. $(P', Q') \in \mathbb{S}$. We write $P \lesssim Q$ whenever $(P, Q) \in \mathbb{S}$ for a (typed) simulation $\mathbb{S}$.*

In the LTS semantics of HCP, the processes P and Q from Theorem 47 are bisimilar; however, in a reduction semantics, any transitions of P that use delayed actions or self-synchronization would be considered as deadlocks, and then Q would only simulate P. Formally:

▶ **Theorem 52.** *The relation $\mathbb{S} = \{(P, R) \mid P \in \mathbb{H}, \text{and } P \rightsquigarrow R\}$ is a simulation.*

Proof sketch. By induction in the structure of P, with a case analysis in $\rightsquigarrow$. See [19] for details. ◀

Finally, Corollary 53, given below, brings Theorem 47 to SP: it shows that even though in HCP both $\mathcal{D}_{\mathbf{H}}(\cdot)$ and $\mathcal{D}^{\mathsf{R}}_{\mathbf{H}}(\cdot)$ are equivalent, in SP those notions are related via a simulation.

▶ **Corollary 53.** *Given $P \in \mathbb{H}_{\mathsf{F}}$, then there exist $Q \in \mathbb{H}$ and $P' \in \mathbb{H}_{\mathsf{R}}$ s.t.: (1) $[\![Q]\!] \in \textsc{Dist}([\![P]\!])$, (2) $[\![Q]\!] \simeq^*_{cc} [\![P']\!]$, (3) $P \lesssim P'$, (4) $[\![P]\!] \approx_{\mathtt{H}} [\![P']\!]$.*

Proof. It follows by Lemma 48 and Theorems 47 and 52. ◀

Table 1 Inventory of type systems, classes of processes, and encodings considered in the paper. Shaded cells concern elements used in the case of asynchronous processes (§ 4).

SP	Definition 2	$\mathbb{C}$	Definition 34	$\llbracket \cdot \rrbracket$: SP → (H)CP	Definition 28
CP	Definition 11	$\mathbb{H}$	Definition 34		
HCP	Definition 12	$\mathbb{H}_{\mathsf{F}}$, $\mathbb{H}_{\mathsf{R}}$	Definition 37		
P	Definition 58	$\mathbb{P}$	Definition 67	$\{\!\!\{ \cdot \}\!\!\}^f$: SP → P	Definition 65
μP	Definition 62	$\mu\mathbb{P}$	Definition 67	$\langle\!\langle \cdot \rangle\!\rangle$: SP$^{\mathfrak{a}}$ → SP	Definition 82
SP$^{\mathfrak{a}}$	Definition 78	$\mathbb{C}^{\mathfrak{a}}$, $\mathbb{H}^{\mathfrak{a}}$	Definition 86		

▶ **Example 54** (Corollary 53, At Work)**.** Consider the process P_{54} and its encoding into HCP:

$$P_{54} \triangleq (\nu\, wz)(\overline{w}\langle\,\rangle.(\nu\, xy)(\overline{x}\langle\,\rangle.\mathbf{0} \mid (\nu\, ku)(k().\mathbf{0} \mid y().z().\overline{u}\langle\,\rangle.\mathbf{0})))$$

$$\llbracket P_{54} \rrbracket \triangleq (\nu\, wz)(w[\,].(\nu\, xy)(x[\,].\mathbf{0} \mid (\nu\, ku)(k().\mathbf{0} \mid y().z().u[\,].\mathbf{0})))$$

Note that $\neg\mathcal{D}_{\mathbf{S}}(P_{54})$ but $\llbracket P_{54} \rrbracket$ has two τ-transitions:

$$\llbracket P_{54} \rrbracket \xrightarrow{\tau} (\nu\, xy)(x[\,].\mathbf{0} \mid (\nu\, ku)(k().\mathbf{0} \mid y().u[\,].\mathbf{0})) \to^{*} \equiv_{\mathtt{H}} \mathbf{0} \qquad (1)$$

$$\llbracket P_{54} \rrbracket \xrightarrow{\tau} (\nu\, wz)(w[\,].(\nu\, ku)(k().\mathbf{0} \mid z().u[\,].\mathbf{0})) \to^{*} \equiv_{\mathtt{H}} \mathbf{0} \qquad (2)$$

where the transition in (1) is due to a delayed action on z and a self-synchronization along the session wz, and the transition in (2) is due to a delayed τ-transition along the session xy. Hence, $\mathcal{D}_{\mathbf{H}}(\llbracket P_{54} \rrbracket)$. Via disentanglement we obtain:

$$\llbracket P_{54} \rrbracket \rightsquigarrow (\nu\, wz)(w[\,].\mathbf{0} \mid (\nu\, xy)(x[\,].\mathbf{0} \mid (\nu\, ku)(k().\mathbf{0} \mid y().z().u[\,].\mathbf{0}))) \triangleq R'$$

In R', neither a delayed action nor a self-synchronization are needed, thus $\mathcal{D}^{\mathsf{R}}_{\mathbf{H}}(R')$. We could also give priority to the τ-transition along wz; however, because the input on z is not at the top-level, a synchronization on wz still requires a delayed action. This shows the need for commuting conversions. Indeed, by using $\simeq_{cc}$ we obtain:

$$R' \simeq_{cc} (\nu\, wz)(w[\,].\mathbf{0} \mid (\nu\, xy)(x[\,].\mathbf{0} \mid (\nu\, ku)(k().\mathbf{0} \mid z().y().u[\,].\mathbf{0}))) \triangleq R$$

We have that $\mathcal{D}^{\mathsf{R}}_{\mathbf{H}}(R)$. Moreover, there is a process P' in SP s.t. $\llbracket P' \rrbracket = R$:

$$P' \triangleq (\nu\, wz)(\overline{w}\langle\,\rangle.\mathbf{0} \mid (\nu\, xy)(\overline{x}\langle\,\rangle.\mathbf{0} \mid (\nu\, ku)(k().\mathbf{0} \mid z().y().\overline{u}\langle\,\rangle.\mathbf{0})))$$

Notice that $P_{54} \lesssim P'$ follows trivially since $\mathcal{D}_{\mathbf{S}}(P')$.

4 The Case of Asynchronous Processes

Up to here we have addressed **(Q1)** by considering the relation between $\mathbb{H}_{\mathsf{F}}$ and $\mathbb{H}_{\mathsf{R}}$ in HCP. We now turn our attention to **(Q2)** and consider the interplay of DF and *asynchronous processes* as typable by Padovani's type system for asynchronous processes, here dubbed P [27].

We proceed as follows. In § 4.1 we recall Padovani's type system and introduce $\mathbb{P}$, the class of SP processes induced by P, and its sub-class $\mu\mathbb{P}$, derived from the type system μP. In § 4.2, we show that $\mu\mathbb{P} = \mathbb{C}$ (Corollary 74). In § 4.3, we establish a key result: the precise relations between $\mathbb{H}$, $\mathbb{H}_{\mathsf{F}}$, $\mathbb{H}_{\mathsf{R}}$, and $\mathbb{P}$ (Theorem 76). Also, we revisit Corollary 53 from the standpoint of P: we prove that the process $P' \in \mathbb{H}_{\mathsf{R}}$ (obtained via commuting conversions and disentanglement) is also in $\mathbb{P}$ (Corollary 77). In § 4.4 we present SP$^{\mathfrak{a}}$, an asynchronous version of SP, and show how to extend our results for SP to SP$^{\mathfrak{a}}$ (Corollary 88).

Table 1 summarizes all technical ingredients, including those introduced in this section.

4.1 The Type System P

We summarize the linear fragment of the type system P by Padovani [27], which streamlines Kobayashi's type system for deadlock freedom [21]. We assume sets of channels $a, b, \ldots$ and variables $x, y, \ldots$; then, names $u, v, \ldots$ are either channels or variables. We find it convenient to use processes with polyadic communication:

$$P ::= u!\langle x_1, \ldots, x_n\rangle \mid u?(x_1, \ldots, x_n).P \mid P \mid Q \mid (\nu\, a)P \mid \mathbf{0}$$

Two salient features are the output $u!\langle x_1, \ldots, x_n\rangle$, which does not have a continuation, and "$(\nu\, a)P$", the standard restriction construct of the π-calculus. Notions of free and bound names are as usual: in the process $u?(x_1, \ldots, x_n).P$, $x_1, \ldots, x_n$ are bound in P; in $(\nu\, a)P$, a is bound in P. We write $\mathsf{fn}(P)$ and $\mathsf{bn}(P)$ for the free and bound names of P, respectively.

The reduction semantics is closed under a structural congruence, denoted $\equiv_{\mathsf{P}}$, which is standardly defined by the following axioms: (i) $P \mid Q \equiv_{\mathsf{P}} Q \mid P$, (ii) $P \mid \mathbf{0} \equiv_{\mathsf{P}} P$, (iii) $P \mid (Q \mid R) \equiv_{\mathsf{P}} (P \mid Q) \mid R$, (iv) $(\nu\, x)P \mid Q \equiv_{\mathsf{P}} (\nu\, x)(P \mid Q)$ (if $x \notin \mathsf{fn}(Q)$), and (v) $(\nu\, x)(\nu\, y) \equiv_{\mathsf{P}} (\nu\, y)(\nu\, x)P$.

Reduction is then defined as

$$a!\langle v_1, \ldots, v_n\rangle \mid a?(x_1, \ldots, x_n).P \longrightarrow_{\mathsf{P}} P\{{}^{v_1, \ldots, v_n}\!/\!{}_{x_1, \ldots, x_n}\}$$

▶ **Example 55.** Differently from SP, channels in Padovani's framework are used exactly once. Structured communications can of course be still expressed, using continuations. Consider the process $P_{55} \triangleq (\nu\, x)(Q_{55} \mid R_{55})$ with

$$Q_{55} \triangleq (\nu\, a)(\nu\, c)(x!\langle a, c\rangle \mid c!\langle a\rangle) \qquad R_{55} \triangleq x?(z_1, w).w?(z_2).((\nu\, r)z_1!\langle r\rangle \mid z_2?(w).\mathbf{0})$$

Q_{55} sends the two endpoints of a channel: the first is sent along x (together with a continuation channel c), and the other is sent along c. Channel a occurs in both outputs, but these occurrences actually denote different endpoints; this difference will be justified by typing (cf. Example 59). R_{55} receives these endpoints to implement a local exchange of name r along a.

Padovani's type system ensures the *absence of pending communications* in a process with respect to a channel. To formalize this notion, we rely on reduction contexts, defined as $\mathcal{C} ::= [\circ] \mid (\mathcal{C} \mid P) \mid (\nu\, x)\mathcal{C}$. We may now define the following auxiliary predicates:

$$\begin{aligned} &\mathsf{out}(a, P) \;\Leftrightarrow\; P \equiv_{\mathsf{P}} \mathcal{C}[a!\langle e\rangle] \,\wedge\, a \notin \mathsf{bn}(C) \qquad \mathsf{in}(a, P) \;\Leftrightarrow\; P \equiv_{\mathsf{P}} \mathcal{C}[a?(x).Q] \,\wedge\, a \notin \mathsf{bn}(C)\\ &\mathsf{sync}(a, P) \;\Leftrightarrow\; (\mathsf{in}(a, P) \wedge \mathsf{out}(a, P)) \end{aligned}$$

Intuitively, predicate $\mathsf{in}(a, P)$ holds if a is free in P and there is a subprocess of P that can perform a linear input on a. Predicate $\mathsf{out}(a, P)$ holds if a is free in P and a subprocess of P is waiting to send a value v. Predicate $\mathsf{sync}(a, P)$ denotes a pending input/output on a for which a synchronization on a is immediately available. This way, to formalize a pending input/output for which a synchronization on a is not immediately possible, we define:

$$\mathsf{wait}(a, P) \;\Leftrightarrow\; (\mathsf{in}(a, P) \;\vee\; \mathsf{out}(a, P)) \;\wedge\; \neg\mathsf{sync}(a, P)$$

Thus, ensuring no pending communications in P means ensuring $\neg\mathsf{wait}(a, P)$ for every a.

▶ **Example 56** (Predicates at work)**.** Consider the process $P_{56} \triangleq a?(x).b!\langle v_1\rangle \mid b?(y).a!\langle v_2\rangle$, which has a circular dependency. We have: (1) $\mathsf{out}(a, P_{56})$ and $\mathsf{out}(b, P_{56})$. (2) $\mathsf{in}(a, P_{56})$ and $\mathsf{in}(b, P_{56})$. (3) $\neg\mathsf{sync}(a, P_{56})$ and $\neg\mathsf{sync}(b, P_{56})$. (4) $\mathsf{wait}(a, P_{56})$ and $\mathsf{wait}(b, P_{56})$.

Type System. To ensure absence of pending communications, P relies on *priorities*. Let $p, q, \ldots$ denote subsets of $\{?, !\}$ (*polarities*). Types for channels are defined as follows:

$$t ::= p[t_1, \ldots, t_m]^n$$

where $n \in \mathbb{Z}$ is the priority of a channel, which indicates urgency for its use: the lower the number, the higher the urgency. We write $\#$ for $\{?, !\}$. This way, $\#$ and $\emptyset$ are even polarities.

▶ **Example 57** (Priorities)**.** To illustrate types and priorities, recall again the process $P_{56} = a?(x).b!\langle v_1\rangle \mid b?(y).a!\langle v_2\rangle$. As we have seen, we have $\mathsf{wait}(a, P_{56})$ and $\mathsf{wait}(b, P_{56})$.

Let us assume that x, y, v_1, v_2 are all typed with some type t. By inspecting the left thread of P_{56}, we infer that a and b must be typed as $?[t]^m$ and $![t]^n$, respectively, for some m, n. Then, by inspecting the right thread, we infer that a and b must then be typed with $![t]^m$ and $?[t]^n$, respectively. This way, their types in the composed process are $\#[t]^m$ (for a) and $\#[t]^n$ (for b). Looking again at the left thread, as the input along a is blocking the output along b, we must have $m < n$; in the right thread we have the exact opposite situation, thus $n < m$. These unsolvable inequalities reveal the circular dependency between a and b, which will make P_{56} not typable in the type system P.

We now summarize how typing excludes pending communications. Type environments $\Gamma, \Gamma', \ldots$ are finite maps from channels to types. We write $dom(\Gamma)$ for the domain of Γ. We write Γ, Γ' for the union of Γ and Γ' when $dom(\Gamma) \cap dom(\Gamma') = \emptyset$. We write $+$ to denote a partial composition operator on types defined as: $p[s_1, \ldots, s_m]^n + q[s_1, \ldots, s_m]^n = (p \cup q)[s_1, \ldots, s_m]^n$ if $p \cap q = \emptyset$, and undefined otherwise. For type environments we have: $\Gamma + \Gamma' \triangleq \Gamma, \Gamma'$ if $dom(\Gamma) \cap dom(\Gamma') = \emptyset$, and $(\Gamma, u : t) + (\Gamma', u : s) \triangleq (\Gamma + \Gamma'), u : t + s$. Let $\mathbb{Z}_\top = \mathbb{Z} \cup \{\top\}$, where $n < \top$ for every $n \in \mathbb{Z}$. The functions $|\cdot|$, from types to $\mathbb{Z}_\top$, and $\$^h$ from types to types are defined as follows:

$$|t| \triangleq \begin{cases} n & \text{if } t = p[s]^n \text{and } p \neq \emptyset \\ \top & \text{otherwise} \end{cases} \qquad \$^h t \triangleq \begin{cases} p[s]^{n+h} & \text{if } t = p[s]^n \text{and } p \neq \emptyset \\ t & \text{otherwise} \end{cases}$$

Intuitively, the function $\$^h$ allows us to shift priorities. We say that a type t is *unlimited*, written $\mathsf{un}(t)$, iff $|t| = \top$. We write $\mathsf{un}(\Gamma)$ iff all the types in the range of Γ are unlimited. We are now ready to give the typing rules, considering only processes with biadic communication, for simplicity. The rules for monadic communication are as expected.

▶ **Definition 58** (P)**.** *The typing rules for processes use judgements of the form* $\Gamma \vdash_{\mathsf{P}} P$ *and* $\Gamma \vdash_{\mathsf{P}} \widetilde{e} : \widetilde{t}$ *and are presented in Figure 9. We shall write* $P \in \mathsf{P}$ *if* $\Gamma \vdash_{\mathsf{P}} P$*, for some* Γ*.*

Rule T-In is used to type inputs $u?(x_1, x_2).P$, where u has type $?[t_1, t_2]^n$. The condition $n < |\Gamma|$ ensures that u has been assigned the highest urgency; accordingly, the types of x and y are shifted by n. Rule T-Out, used to type outputs on channels of type $![t_1, t_2]^n$, follows a similar principle: the types of x and y are shifted by n, since they must have lower urgency than u. Rule T-New is used for restricting new channels with full ($\#$) or empty ($\emptyset$) polarity.

▶ **Example 59.** Recall processes Q_{55} and R_{55} from Example 55. Consider types $s' = \$^1![\emptyset[]^4]^2$ and $s = \$^2?[\emptyset[]^4]^1$. Note that $s' + s = \#[\emptyset[]^4]^3$. Process Q_{55} is typed as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{a : s', c : \$^1?[s]^1 \vdash_{\mathsf{P}} a : s', c : \$^1?[s]^1}{a : s', c : \$^1?[s]^1, x :![s', \$^1?[s]^1]^1 \vdash_{\mathsf{P}} x!\langle a, c\rangle}\,\text{T-Out} \qquad \dfrac{a : s \vdash_{\mathsf{P}} a : s}{a : s, c :![s]^2 \vdash_{\mathsf{P}} c!\langle a\rangle}\,\text{T-Out}}{a : \#[\emptyset[]^4]^3, x :![s', \$^1?[s]^1]^1, c : \#[s]^2 \vdash_{\mathsf{P}} x!\langle a, c\rangle \mid c!\langle a\rangle}\,\text{T-Par}}{a : \#[\emptyset[]^4]^3, x :![s', \$^1?[s]^1]^1 \vdash_{\mathsf{P}} (\nu\, c)(x!\langle a, c\rangle \mid c!\langle a\rangle)}\,\text{T-New}}{x :![s', \$^1?[s]^1]^1 \vdash_{\mathsf{P}} (\nu\, a)(\nu\, c)(x!\langle a, c\rangle \mid c!\langle a\rangle)}\,\text{T-New}$$

$$\textsc{T-Idle}\ \frac{\mathsf{un}(\Gamma)}{\Gamma \vdash_{\mathsf{P}} \mathbf{0}} \qquad \textsc{T-Par}\ \frac{\Gamma_1 \vdash_{\mathsf{P}} P_1 \quad \Gamma_2 \vdash_{\mathsf{P}} P_2}{\Gamma_1 + \Gamma_2 \vdash_{\mathsf{P}} P_1 \mid P_2} \qquad \textsc{T-out}\ \frac{\Gamma, e_1 : \$^n t_1, e_2 : \$^n t_2 \vdash_{\mathsf{P}} e_1 : \$^n t_1, e_2 : \$^n t_2 \quad \mathsf{un}(\Gamma)}{\Gamma + u :![t_1, t_2]^n \vdash_{\mathsf{P}} u!\langle e_1, e_2\rangle}$$

$$\textsc{T-In}\ \frac{\Gamma, x : \$^n t_1, y : \$^n t_2 \vdash_{\mathsf{P}} P \qquad n < |\Gamma|}{\Gamma + u :?[t_1, t_2]^n \vdash_{\mathsf{P}} u?(x,y).P} \qquad \textsc{T-New}\ \frac{\Gamma, a : p[t]^n \vdash_{\mathsf{P}} P \qquad p \text{ even}}{\Gamma \vdash_{\mathsf{P}} (\nu\, a)P}$$

Figure 9 P: Typing rules for asynchronous processes.

$$\textsc{M-Idle}\ \frac{\mathsf{un}(\Gamma)}{\Gamma \vdash^{\mu}_{\mathsf{P}} \mathbf{0}} \qquad \textsc{M-Out}\ \frac{\Gamma, e_1 : \$^n t_1, e_2 : \$^n t_2 \vdash^{\mu}_{\mathsf{P}} e_1 : \$^n t_1, e_2 : \$^n t_2 \quad \mathsf{un}(\Gamma)}{\Gamma + u :![t_1, t_2]^n \vdash^{\mu}_{\mathsf{P}} u!\langle e_1, e_2\rangle}$$

$$\textsc{M-In}\ \frac{\Gamma, x : \$^n t_1, y : \$^n t_2 \vdash^{\mu}_{\mathsf{P}} P \qquad n < |\Gamma|}{\Gamma + u :?[t_1, t_2]^n \vdash^{\mu}_{\mathsf{P}} u?(x,y).P}$$

$$\textsc{M-New-Par}\ \frac{\Gamma_1, a : p[t]^n \vdash^{\mu}_{\mathsf{P}} P \qquad \Gamma_2, a : q[t]^n \vdash^{\mu}_{\mathsf{P}} Q \qquad dom(\Gamma_1) \cap dom(\Gamma_2) = \emptyset,\ p \cup q \text{ even}}{\Gamma_1 + \Gamma_2 \vdash^{\mu}_{\mathsf{P}} (\nu\, a)(P \mid Q)}$$

Figure 10 μP: Modified typing rules for asynchronous processes.

Similarly we can check that $x :?[s', \$^1?[s]^1]^1 \vdash_{\mathsf{P}} R_{55}$. Note that x is typed with opposite polarities in Q_{55} and R_{55}; thus by applying Rules T-Par and T-New we obtain $\vdash_{\mathsf{P}} P_{55}$.

We may now characterize deadlock freedom. Let us write $\longrightarrow^*_{\mathsf{P}}$ to denote the reflexive, transitive closure of $\longrightarrow_{\mathsf{P}}$. Also, write $P \not\longrightarrow_{\mathsf{P}}$ if there is no Q such that $P \longrightarrow_{\mathsf{P}} Q$. We have:

▶ **Definition 60** (P: Deadlock Freedom [27]). *We say that $P \in \mathsf{P}$ is deadlock free, written $\mathcal{D}_{\mathbf{P}}(P)$, if for every Q such that $P \longrightarrow^*_{\mathsf{P}} (\nu\, \tilde{a})Q \not\longrightarrow_{\mathsf{P}}$ we have $\neg\mathsf{wait}(a, Q)$ for every a.*

We have that processes typed in the empty environment are deadlock free:

▶ **Theorem 61** (P: Deadlock Freedom [27]). *If $\vdash_{\mathsf{P}} P$, then $\mathcal{D}_{\mathbf{P}}(P)$.*

4.2 Relating $\mathbb{P}$ and $\mathbb{C}$

We now consider μP ("micro P"), which is a fragment of P that is similar to CP. We also define the encoding $\{\!\![\cdot]\!\!\}^f : \mathsf{SP} \to \mathsf{P}$, and $\mathbb{P}$ and $\mu\mathbb{P}$ the classes of SP processes that are captured by P and μP, respectively. These are all ingredients for our results in § 4.3.

▶ **Definition 62** (μP). *The typing rules for processes are presented in Figure 10. A judgement $\Gamma \vdash^{\mu}_{\mathsf{P}} P$ denotes that P is well typed in Γ. We shall write $P \in \mu\mathsf{P}$ if $\Gamma \vdash^{\mu}_{\mathsf{P}} P$, for some Γ.*

μP results from P by replacing Rules T-Par and T-New in Figure 9 with Rule M-New-Par, which combines restriction and parallel just as Rule C-Cut in Figure 4.

▶ **Lemma 63.** *If $\Gamma \vdash^{\mu}_{\mathsf{P}} P$ then $\Gamma \vdash_{\mathsf{P}} P$.*

Proof sketch. By induction on the type derivation, with a case analysis in the last rule applied. See [19] for details. ◀

▶ **Corollary 64.** *If $\vdash^{\mu}_{\mathsf{P}} P$, then $\mathcal{D}_{\mathbf{P}}(P)$.*

Proof. It follows by Lemma 63 and Theorem 61. ◀

We now define an encoding from SP to P, following a continuation-passing style (cf. [8]). Let f be a function from names to names. We use the following conventions: f_x stands for $f(x)$, and $f, \{x \mapsto c\}$ stands for the function f' s.t. $f'(x) = c$ and $f'(y) = f(y)$ if $x \neq y$.

▶ **Definition 65.** *The (partial) encoding $\{\!\![\cdot]\!\!\}^f : \mathsf{SP} \to \mathsf{P}$ is parameterized by a function, ranging over f, g, from names to names, and is defined for processes as follows:*

$$\begin{aligned}
\{\!\![\overline{x}\langle v\rangle.P]\!\!\}^f &\triangleq (\nu\, m)(f_x!\langle f_v, m\rangle \mid m?(k).(\nu\, c)(k!\langle c\rangle \mid \{\!\![P]\!\!\}^{f,\{x\mapsto c\}}))\\
\{\!\![x(z).P]\!\!\}^f &\triangleq f_x?(z,m).(\nu\, a)(m!\langle a\rangle \mid a?(c).\{\!\![P]\!\!\}^{f,\{x\mapsto c\}})\\
\{\!\![\overline{x}\langle\,\rangle.\mathbf{0}]\!\!\}^f &\triangleq (\nu\, r) f_x!\langle r\rangle\\
\{\!\![x().P]\!\!\}^f &\triangleq f_x?(r).\{\!\![P]\!\!\}^f \qquad (\textit{with } r \notin \mathtt{fv}(P) \cup \mathtt{bv}(P))\\
\{\!\![(\nu\, xy)P]\!\!\}^f &\triangleq (\nu\, d)\{\!\![P]\!\!\}^{f,\{x,y\mapsto d\}}
\end{aligned}$$

and as a homomorphism for $P \mid Q$ and $\mathbf{0}$. The encoding of session types is as follows:

$$\begin{aligned}
\{\!\![\mathsf{end}_?]\!\!\} &\triangleq ?[\emptyset[]]^n & \qquad \{\!\![!S.T]\!\!\} &\triangleq ?[\{\!\![T]\!\!\}, ![![\{\!\![S]\!\!\}]^i]^j]^n\\
\{\!\![\mathsf{end}_!]\!\!\} &\triangleq ![\emptyset[]]^n & \qquad \{\!\![?S.T]\!\!\} &\triangleq ![\{\!\![T]\!\!\}, ![![\{\!\![\overline{S}]\!\!\}]^i]^j]^n
\end{aligned}$$

Typing environments are encoded inductively as $\{\!\![\emptyset]\!\!\}^f \triangleq \emptyset$ and $\{\!\![\Gamma, x : T]\!\!\}^f \triangleq \{\!\![\Gamma]\!\!\}^f, f_x : \{\!\![T]\!\!\}$. The values of i, j and n are calculated when checking in μP the typablity of $\{\!\![\Gamma]\!\!\}^f \vdash^{\mu}_{\mathsf{P}} \{\!\![P]\!\!\}^f$.

Intuitively, the encoding of processes in Definition 65 collapses the two endpoints of a session into a single name in P. The encoding of $\overline{x}\langle v\rangle.P$ mimics both the sequential structure of a session and synchronous communication. $\{\!\![\overline{x}\langle v\rangle.P]\!\!\}^f$ sends the (renamed) value f_v and a new channel m along which an input $m?(k)$. blocks the continuation $\{\!\![P]\!\!\}^{f,\{x\mapsto c\}}$, and receives the channel along which the the continuation c of the session x will be sent. Dually, $\{\!\![x(z).P]\!\!\}^f$ receives the intended value and the channel m, which sends the channel a, along which the continuation of the session x will be received. Finally, the encoding is partial because $\{\!\![\overline{x}\langle\,\rangle.P]\!\!\}^f$ is only defined when $P = \mathbf{0}$: this allows us to characterize the session processes captured in CP, where the empty output has no continuation (see Definition 11).

▶ **Example 66.** Consider the process $P_{66} \triangleq (\nu\, xy)(\overline{x}\langle v\rangle.R \mid y(z).Q)$ in SP. We show $\{\!\![P_{66}]\!\!\}^f$ and illustrate its reductions. Let us assume P_{66} is well-typed. Because of linearity of x and y, they only appear in the left and right subprocess, respectively. Let $f = \{x, y \mapsto d\}$ and $g_1 = \{x \mapsto c\}$,$g_2 = \{y \mapsto c'\}$, then we have:

$$\begin{aligned}
\{\!\![P_{66}]\!\!\}^{\emptyset} &= (\nu\, d)(\{\!\![\overline{x}\langle v\rangle.R]\!\!\}^f \mid \{\!\![y(z).Q]\!\!\}^f)\\
&= (\nu\, d)((\nu\, m)(d!\langle v, m\rangle \mid m?(k).(\nu\, c)(k!\langle c\rangle \mid \{\!\![R]\!\!\}^{g_1}))\\
&\qquad\qquad \mid d?(z,m').(\nu\, a)(m'!\langle a\rangle \mid a?(c').\{\!\![Q]\!\!\}^{g_2}))\\
&\longrightarrow_{\mathsf{P}} (\nu\, m)(m?(k).(\nu\, c)(k!\langle c\rangle \mid \{\!\![R]\!\!\}^{g_1}) \mid (\nu\, a)(m!\langle a\rangle \mid a?(c').\{\!\![Q]\!\!\}^{g_2}\{v/z\}))\\
&\longrightarrow_{\mathsf{P}} (\nu\, a)((\nu\, c)(a!\langle c\rangle \mid \{\!\![R]\!\!\}^{g_1}) \mid a?(c').\{\!\![Q]\!\!\}^{g_2}\{v/z\})\\
&\longrightarrow_{\mathsf{P}} (\nu\, c)(\{\!\![R]\!\!\}^{g_1} \mid \{\!\![Q]\!\!\}^{g_2}\{v/z, c/c'\})
\end{aligned}$$

Our main result concerns the classes of processes associated to P and $\mu\mathsf{P}$, which we define by mirroring the definition of $\mathbb{C}$ and $\mathbb{H}$ (Definition 34):

▶ **Definition 67** ($\mathbb{P}$ and $\mu\mathbb{P}$). *The classes of processes $\mathbb{P}$ and $\mu\mathbb{P}$ are defined as follows:*

$$\mathbb{P} \triangleq \{P \in \mathsf{SP} \mid \Gamma \vdash_{\mathsf{S}} P \wedge \{\![\Gamma]\!\}^f \vdash_{\mathsf{P}} \{\![P]\!\}^f\} \qquad \mu\mathbb{P} \triangleq \{P \in \mathsf{SP} \mid \Gamma \vdash_{\mathsf{S}} P \wedge \{\![\Gamma]\!\}^f \vdash^{\mu}_{\mathsf{P}} \{\![P]\!\}^f\}$$

The following result establishes an operational correspondence between a process and its encoding, what will allows us to capture deadlock free session processes via the encoding.

▶ **Theorem 68** (Operational Correspondence for $\{\![\cdot]\!\}^f$). *Let $P \in \mu\mathbb{P}$. We have:*

1. **a.** *If $P \equiv (\nu\, \widetilde{xy})(\overline{x}\langle\,\rangle.Q_1 \mid y().Q_2 \mid S)$, then $\exists Q \in \mathsf{P}$ such that $\{\![P]\!\}^f \longrightarrow_{\mathsf{P}} Q$ and $Q \equiv_{\mathsf{P}} \{\![(\nu\, \widetilde{xy})(Q_1 \mid Q_2 \mid S)]\!\}^f$.*
   b. *If $P \equiv (\nu\, \widetilde{xy})(\overline{x}\langle v\rangle.Q_1 \mid y(z).Q_2 \mid S)$, then $\exists Q \in \mathsf{P}$ such that $\{\![P]\!\}^f \longrightarrow^3_{\mathsf{P}} Q$ and $Q \equiv_{\mathsf{P}} \{\![(\nu\, \widetilde{xy})(Q_1 \mid Q_2\{v/z\} \mid S)]\!\}^f$.*
2. *If $\{\![P]\!\}^f \longrightarrow_{\mathsf{P}} R$, then either:*
   a. *$\exists P' \in \mathsf{SP}$ such that $P \longrightarrow P'$ and $R \equiv_{\mathsf{P}} \{\![P']\!\}^f$, or*
   b. *$\exists Q \in \mathsf{P}$, and $P' \in \mathsf{SP}$ such that $R \longrightarrow^2_{\mathsf{P}} Q$, $P \longrightarrow P'$ and $Q \equiv_{\mathsf{P}} \{\![P']\!\}^f$.*

Proof. By assumption, $P \in \mu\mathbb{P}$ and so we have that $\Gamma \vdash_{\mathsf{S}} P$, for some Γ, and $\{\![\Gamma]\!\}^f \vdash^{\mu}_{\mathsf{P}} \{\![P]\!\}^f$ is defined. Moreover, by Corollary 8 P is well-formed. We consider each item separately. Item 1(b) follows by Definition 65 and the rest of the analysis proceeds as in Example 66; Item 1(a) is similar. Item (2): If $\{\![P]\!\}^f \longrightarrow_{\mathsf{P}} R$, then $\{\![P]\!\}^f \equiv_{\mathsf{P}} (\nu\, \tilde{a})(d!\langle\tilde{v}\rangle \mid d?(\tilde{x}).K \mid S)$. By typability of P and Definition 65 one of the two possibilities apply:

i $P \equiv (\nu\, \widetilde{xy})(\overline{x}\langle\,\rangle.\mathbf{0} \mid y().K' \mid S')$, $\{\![S']\!\}^f = S$, $\{\![K']\!\}^f = K$ and $f = f', \{x, y \mapsto d\}$,

ii $P \equiv (\nu\, \widetilde{xy})(\overline{x}\langle v\rangle.S_1 \mid y(z).K' \mid S_2)$ and $f = f', \{x, y \mapsto d\}$

In case (i) $\{\![P]\!\}^f \equiv_{\mathsf{P}} (\nu\, \tilde{a})((\nu\, r)d!\langle r\rangle \mid d?(r').K \mid S) \longrightarrow_{\mathsf{P}} (\nu\, \tilde{a})(K \mid S)$ and $P \equiv (\nu\, \widetilde{xy})(\overline{x}\langle\,\rangle.\mathbf{0} \mid y().K' \mid S') \longrightarrow (\nu\, \widetilde{xy}')(K' \mid S'')$, which verifies case (2a). In case (ii) we proceed as in Example 66, which verifies case (2b). ◀

As a consequence of Theorem 68 we have the following result, which shows the relation between deadlock-free processes in $\mu\mathbb{P}$.

▶ **Corollary 69.** *Let $P \in \mu\mathbb{P}$. $\mathcal{D}_{\mathbf{S}}(P)$ iff $\mathcal{D}_{\mathbf{P}}(\{\![P]\!\}^f)$.*

Proof. For each direction we prove its contrapositive. We discuss only the right-to-left direction; the analysis for the left-to-right direction is similar. By Definition 60, there exist R and a s.t. $\{\![P]\!\}^f \longrightarrow^*_{\mathsf{P}} (\nu\, \tilde{a})R \not\longrightarrow_{\mathsf{P}}$ and $\mathsf{wait}(a, R)$. If $\mathsf{wait}(a, R)$ holds because $\mathsf{in}(a, R)$ then $R \equiv_{\mathsf{P}} (\nu\, \tilde{a})(a?(x).Q \mid S)$. By Theorem 68 and Definition 65 there exists $P' \equiv (\nu\, \widetilde{xy})(z(w).Q' \mid S')$ s.t. $P \longrightarrow^* P'$ and $R \equiv_{\mathsf{P}} \{\![P']\!\}^f$. Note that $P' \not\longrightarrow$, since otherwise by Theorem 68(1), $(\nu\, \tilde{a})R \longrightarrow_{\mathsf{P}} R''$ for some process R'', contradicting our assumption, thus $\neg\mathcal{D}_{\mathbf{S}}(P)$. The case when $\mathsf{wait}(a, R)$ holds because $\mathsf{out}(a, R)$ proceeds similarly. ◀

Corollary 69 allows us to detect DF session processes via the encoding.

4.3 Comparing $\mathbb{H}$ and $\mathbb{P}$

From § 3 we infer that a process $P \in \mathbb{H}$, typed as $[\![P]\!] \vdash_{\mathsf{H}} \Gamma_1 \parallel \cdots \parallel \Gamma_n$, can be decomposed into processes $P_1 \vdash_{\mathsf{C}} \Gamma_1, \ldots, P_n \vdash_{\mathsf{C}} \Gamma_n$. Moreover, by Lemma 48 (disentanglement in SP) we infer that processes $P_1, \ldots, P_n$ have a pre-image in SP, therefore $\forall i \in [1, n].\, P_i \in \mathbb{C} \subsetneq \mathbb{H}$.

As an important preliminary step, we first prove that $\mathbb{C} = \mu\mathbb{P}$ (Corollary 74). We split the proof in two separate inclusions. The following lemma is useful to prove that $\mathbb{C} \subseteq \mu\mathbb{P}$.

▶ **Lemma 70.** *Let Γ and P be an environment and a process in P, resp. The following hold:*

1. *If $\Gamma \vdash_{\mathtt{P}} P$, then $\$^n\Gamma \vdash_{\mathtt{P}} P$.*
2. *If $\Gamma \vdash^{\mu}_{\mathtt{P}} P$, then $\$^n\Gamma \vdash^{\mu}_{\mathtt{P}} P$.*
3. *Let $P \in \mathsf{P}$. If $\Gamma \vdash^{\mu}_{\mathtt{P}} P$ and $\mathsf{un}(T)$, then $\Gamma, x : T \vdash^{\mu}_{\mathtt{P}} P$, with x fresh wrt P.*
4. *Let $\Gamma_1, \dots, \Gamma_n$ be typing environments in SP, then $\{\!\!\{\Gamma_1, \cdots, \Gamma_n\}\!\!\}^f = \{\!\!\{\Gamma_1\}\!\!\}^f + \cdots + \{\!\!\{\Gamma_n\}\!\!\}^f$, for some renaming function f.*
5. *Let T be a session type. Then $\{\!\!\{T\}\!\!\}^f + \$^m\{\!\!\{\overline{T}\}\!\!\}^f = p[T']^n$, for some $m \geq 0$, $n \in \mathbb{Z}$, T' a type in P, and p even.*

Proof. (1) Proven in [27]. (2) Follows from Lemma 63 and item 1. (3) Follows by induction on the type derivation. (Figure 10). (4) Follows immediately by linearity of names in Γ_i, for $i \in [1, n]$. (5) Follows by induction on the structure of T. ◀

▶ **Theorem 71.** $\mathbb{C} \subseteq \mu\mathbb{P}$.

Proof sketch. By induction on the type derivation $\Gamma \vdash_{\mathtt{S}} P$ with a case analysis in the last rule applied. See [19] for details. ◀

To prove the other direction we need the following result.

▶ **Lemma 72.** *Let S and T be session types. We have: (1) $\{\!\!\{S\}\!\!\}^f + \{\!\!\{T\}\!\!\}^f$ is defined iff $T = \overline{S}$. (2) $[\![\overline{S}]\!] = ([\![S]\!])^{\perp}$.*

Proof. Item (1) follows by Definition 65 and def. of $+$. Item (2) follows immediately by induction in the structure of S and Definition 28. ◀

▶ **Theorem 73.** $\mu\mathbb{P} \subseteq \mathbb{C}$.

Proof sketch. By induction on the type derivation $\Gamma \vdash_{\mathtt{S}} P$ with a case analysis in the last rule applied. See [19] for details. ◀

▶ **Corollary 74.** $\mathbb{C} = \mu\mathbb{P}$.

Proof. It follows by Theorems 71 and 73. ◀

Main Results. We may now return to considering $\mathbb{H}$. Note that $\mathbb{H} \setminus \mathbb{P} \neq \emptyset$: to see this, consider, e.g., the self-synchronizing process $P_1 \in \mathbb{H}_{\mathsf{F}}$ from Example 18. Using Corollary 74 and disentanglement (Definition 42) we show that given a process $P \in \mathbb{H}_{\mathsf{F}}$, we can find a deadlock-free process $P' \in \mathbb{P}$. Formally, we have:

▶ **Theorem 75.** *Given $P \in \mathbb{H}_{\mathsf{F}}$, $\exists P' \in \mathbb{H}_{\mathsf{R}}$ s.t. $P' \in \mathbb{P}$, $[\![P]\!] \approx_{\mathtt{H}} [\![P']\!]$, and $\mathcal{D}_{\mathbf{P}}(\{\!\!\{P'\}\!\!\}^f)$.*

Proof. By Corollary 53, $[\![P]\!] \rightsquigarrow^* \simeq^*_{cc} [\![P']\!] = P_1 \mid \cdots \mid P_n$, and by Lemma 48 $\forall i \in [1, n]$, there exists $P_i' \in \mathbb{C}$ s.t. $[\![P_i']\!] = P_i$. From Corollary 74 and Lemma 63 we infer $\{\!\!\{P_i'\}\!\!\}^f \in \mathbb{P}$, $\forall i \in [1, n]$. Moreover, by repeatedly applying Rule T-Par we have that $P' \in \mathbb{P}$. Finally, $\mathcal{D}_{\mathbf{P}}(\{\!\!\{P'\}\!\!\}^f)$ follows from Theorem 39 and Corollary 69. ◀

The following result relates $\mathbb{H}$, its subclasses $\mathbb{H}_{\mathsf{F}}$ and $\mathbb{H}_{\mathsf{R}}$, and $\mathbb{P}$.

▶ **Theorem 76.** *The following hold:* (1) $\mathbb{P} \setminus \mathbb{H} \neq \emptyset$. (2) $\mathbb{H}_{\mathsf{F}} \setminus \mathbb{P} \neq \emptyset$. (3) $\mathbb{P} \cap \mathbb{H}_{\mathsf{R}} \neq \emptyset$.

Proof sketch. We give witnesses for each statement; see [19] for details. ◀

Finally, from Theorem 75 we have the following result, which complements Corollary 53: given $P \in \mathbb{H}_{\mathsf{F}}$, there exists a bisimilar process $P' \in \mathbb{H}_{\mathsf{R}} \cap \mathbb{P}$ via disentanglement. Note that this result applies in particular to processes in $\mathbb{H}_{\mathsf{F}} \setminus \mathbb{P}$.

▶ **Corollary 77.** *Given $P \in \mathbb{H}_{\mathsf{F}}$, there exists $P' \in \mathbb{H}_{\mathsf{R}}$ s.t.* (1) $P \lesssim P'$, (2) $[\![P]\!] \approx_{\mathtt{H}} [\![P']\!]$, *and* (2) $\mathcal{D}_{\mathbf{P}}(\{\!\!\{P'\}\!\!\}^f)$.

$$\frac{}{v : T, c : S, x :!T.S \vdash^{\mathsf{a}}_{\mathsf{S}} \overline{x}\langle v, c\rangle}\ \textsc{A-Out} \qquad \frac{\Gamma, y : T, w : S \vdash^{\mathsf{a}}_{\mathsf{S}} P}{\Gamma, x :?T.S \vdash^{\mathsf{a}}_{\mathsf{S}} x(y, w).P}\ \textsc{A-In} \qquad \frac{}{\cdot \vdash^{\mathsf{a}}_{\mathsf{S}} \mathbf{0}}\ \textsc{A-Inact}$$

$$\frac{\Gamma \vdash^{\mathsf{a}}_{\mathsf{S}} P}{\Gamma, x : \mathsf{end}_? \vdash^{\mathsf{a}}_{\mathsf{S}} x().P}\ \textsc{A-EInput} \qquad \frac{}{x : \mathsf{end}_! \vdash^{\mathsf{a}}_{\mathsf{S}} \overline{x}\langle\,\rangle}\ \textsc{A-EOutput} \qquad \frac{\Gamma, x : T, y : \overline{T} \vdash^{\mathsf{a}}_{\mathsf{S}} P}{\Gamma \vdash^{\mathsf{a}}_{\mathsf{S}} (\nu\, xy)P}\ \textsc{A-Res} \qquad \frac{\Gamma_1 \vdash^{\mathsf{a}}_{\mathsf{S}} P \quad \Gamma_2 \vdash^{\mathsf{a}}_{\mathsf{S}} Q}{\Gamma_1, \Gamma_2 \vdash^{\mathsf{a}}_{\mathsf{S}} P \mid Q}\ \textsc{A-Par}$$

Figure 11 $\mathsf{SP}^{\mathfrak{a}}$: Typing Rules for Processes.

4.4 Asynchronous Session Processes

Up to here, we have considered asynchronous communication somewhat indirectly: we have examined the type system for asynchronous processes P, defined its associated classes $\mathbb{P}$ and $\mu\mathbb{P}$, and related them to $\mathbb{C}$ and $\mathbb{H}_{\mathsf{F}}$. To treat asynchronous communication directly, here we consider $\mathsf{SP}^{\mathfrak{a}}$, an asynchronous variant of SP, and discuss how to transfer our results from §§ 3 and 4.3 from SP to $\mathsf{SP}^{\mathfrak{a}}$ by leveraging an encoding $\langle\!\langle \cdot \rangle\!\rangle : \mathsf{SP}^{\mathfrak{a}} \to \mathsf{SP}$.

$\mathsf{SP}^{\mathfrak{a}}$: Asynchronous SP. We define $\mathsf{SP}^{\mathfrak{a}}$ as a variant of SP with asynchronous communication. Considerations about variables, endpoints, and values are as for SP (cf. § 2.1). The syntax of processes is largely as before; only constructs for output and input are modified:

$$P \quad ::= \ \mathbf{0} \mid x().P \mid \overline{x}\langle\,\rangle \mid \overline{x}\langle v, c\rangle \mid x(y, w).P \mid P_1 \mid P_2 \mid (\nu\, xy)P$$

Process $\overline{x}\langle v, c\rangle$ sends names v and c along x: the former is a message and the latter is a continuation. Because it has no continuation, this process may be seen as an "output particle" that carries a message. Accordingly, process $x(y, w).P$ receives two values v and c along x and continues as $P\{v/y, c/w\}$, i.e., the process resulting from the capture-avoiding substitution of y by v, and w by c in P. The construct for empty output is also modified. In both $z(x, y).P$ and $(\nu\, xy)P$ x, y are bound in P.

▶ **Definition 78** ($\mathsf{SP}^{\mathfrak{a}}$). *The typing rules for processes are presented in Figure 11. A judgement $\Gamma \vdash^{\mathsf{a}}_{\mathsf{S}} P$ denotes that P is well typed in Γ. We shall write $P \in \mathsf{SP}^{\mathfrak{a}}$ if $\Gamma \vdash^{\mathsf{a}}_{\mathsf{S}} P$, for some Γ.*

Most typing rules are self-explanatory. Rule A-EOutput types an empty output without continuation. Rule A-Out types the output particle $\overline{x}\langle v, c\rangle$ where $x :!T.S$. Since the output has no continuation, the session name c typed as S serves a a continuation name for x. Dually, Rule A-In types x as $?T.S$ provided that y and w are typed as T and S in P, respectively, thus, w serves as a continuation for x in P.

The reduction semantics for $\mathsf{SP}^{\mathfrak{a}}$ is defined as follows:

$$(\nu\, xy)(\overline{x}\langle v, c\rangle \mid y(z, w).Q \ \mid \ S) \longrightarrow (Q\{v/z, c/w\} \mid S) \qquad \textsc{A-Com}$$

$$(\nu\, xy)(\overline{x}\langle\,\rangle \mid y().Q \mid S) \longrightarrow Q \mid S \qquad \textsc{A-EmptyCom}$$

Structural rules and the congruence $\equiv$ are defined as in Figure 2. A labeled reduction semantics $P \xrightarrow{\tau:xy} Q$ for $\mathsf{SP}^{\mathfrak{a}}$ follows easily from Definition 50.

▶ **Example 79.** Consider the process P_{10} from Example 10, which is typable and deadlocked in SP. The following process can be seen as the analogue of P_{10} but in the asynchronous setting of SP^{a}. We define P_{79} as $(\nu\, x_1y_1)(\nu\, x_2y_2)(\nu\, v_1k_1)(\nu\, v_2k_2)P'$, where

$$\begin{aligned}P' \triangleq\ & x_1(z_1, w_1).x_2(z_2, w_2).w_1().w_2().z_1().z_2().\mathbf{0}\\ & \mid (\nu\, c_2c_2')(\overline{y_2}\langle v_2, c_2\rangle \mid \overline{c_2'}\langle\,\rangle) \mid (\nu\, c_1c_1')(\overline{y_1}\langle v_1, c_1\rangle \mid \overline{c_1'}\langle\,\rangle) \mid \overline{k_1}\langle\,\rangle \mid \overline{k_2}\langle\,\rangle\end{aligned}$$

Note how c_1 and c_2 serve as a continuation for y_1 and y_2, respectively. There exist processes Q_1, Q_2 such that $P_{79} \xRightarrow{\tau:x_1y_1} Q_1 \xRightarrow{\tau:x_2y_2} Q_2 \longrightarrow^* \equiv\ \mathbf{0}$.

We define DF for SP^{a} by adapting Definition 9:

▶ **Definition 80** (SP^{a}: Deadlock Freedom)**.** *A process $P \in \mathsf{SP}^{\mathsf{a}}$ is deadlock free, written $\mathcal{D}_{\mathbf{A}}(P)$, if the following holds: whenever $P \longrightarrow^* P'$ and one of the following holds: (1) $P' \equiv (\nu\, \tilde{xy})(\overline{x_i}\langle v, c\rangle \mid Q_2)$, (2) $P' \equiv (\nu\, \tilde{xy})(x_i(y).Q_1 \mid Q_2)$, (3) $P' \equiv (\nu\, \tilde{xy})(x_i().Q_1 \mid Q_2)$, or (4) $P' \equiv (\nu\, \tilde{xy})(\overline{x_i}\langle\,\rangle \mid Q_2)$ (with $x_i \in \tilde{x}$ in all cases) then there exists R such that $P' \longrightarrow R$.*

▶ **Example 81** (Comparing DF in SP and SP^{a})**.** As just illustrated by means of processes P_{10} (from Example 10) and P_{79}, moving to an asynchronous setting may solve certain deadlocks: we have $\neg\mathcal{D}_{\mathbf{S}}(P_{10})$ and $\mathcal{D}_{\mathbf{A}}(P_{79})$.

On the other hand, moving to an asynchronous setting alone does not resolve the problem. Consider the process $Q_{81} \triangleq (\nu\, x_2y_2)(y_2().\mathbf{0} \mid (\nu\, x_1y_1)(y_1().(\overline{x_2}\langle\,\rangle \mid \overline{x_1}\langle\,\rangle)))$. Clearly, Q_{81} is typable in SP^{a}. Also, let Q^*_{81} be the SP variant of Q_{81} with $\mathbf{0}$ as continuation for outputs, i.e., $Q^*_{81} \triangleq (\nu\, x_2y_2)(y_2().\mathbf{0} \mid (\nu\, x_1y_1)(y_1().(\overline{x_2}\langle\,\rangle.\mathbf{0} \mid \overline{x_1}\langle\,\rangle.\mathbf{0})))$. We have that Q^*_{81} is typable in SP and that $\neg\mathcal{D}_{\mathbf{S}}(Q^*_{81})$ and $\neg\mathcal{D}_{\mathbf{A}}(Q_{81})$.

Our goal is to transfer our technical results for SP to the asynchronous setting of SP^{a}. To this end, we define the following encoding:

▶ **Definition 82.** *We define the encoding $\langle\!\langle \cdot \rangle\!\rangle$ from SP^{a} to SP as follows:*

$$\begin{aligned}\langle\!\langle \overline{x}\langle v, c\rangle \rangle\!\rangle &\triangleq \overline{x}\langle v\rangle.\overline{x}\langle c\rangle.\overline{x}\langle\,\rangle.\mathbf{0} & \langle\!\langle \mathsf{end}_? \rangle\!\rangle &\triangleq \mathsf{end}_?\\ \langle\!\langle x(y, w).P \rangle\!\rangle &\triangleq x(y).x(w).x().\langle\!\langle P \rangle\!\rangle & \langle\!\langle \mathsf{end}_! \rangle\!\rangle &\triangleq \mathsf{end}_!\\ \langle\!\langle x().P \rangle\!\rangle &\triangleq x().\langle\!\langle P \rangle\!\rangle & \langle\!\langle !T.S \rangle\!\rangle &\triangleq !T.!S.\mathsf{end}_!\\ \langle\!\langle \overline{x}\langle\,\rangle \rangle\!\rangle &\triangleq \overline{x}\langle\,\rangle.\mathbf{0} & \langle\!\langle ?T.S \rangle\!\rangle &\triangleq ?T.?S.\mathsf{end}_?\end{aligned}$$

and as an homomorphism for $\mathbf{0}$, $P \mid Q$, and $(\nu\, xy)P$.

▶ **Example 83** ($\langle\!\langle \cdot \rangle\!\rangle$, At Work)**.** Consider again the process P_{79}. We have seen that $\mathcal{D}_{\mathbf{A}}(P_{79})$. We now check that $\langle\!\langle \cdot \rangle\!\rangle$ does not add deadlocks by verifying that $\mathcal{D}_{\mathbf{S}}(\langle\!\langle P_{79} \rangle\!\rangle)$:

$$\begin{aligned}\langle\!\langle P_{79} \rangle\!\rangle \triangleq\ & (\nu\, x_1y_1)(\nu\, x_2y_2)(\nu\, v_1k_1)(\nu\, v_2k_2)\langle\!\langle P' \rangle\!\rangle\\ \langle\!\langle P' \rangle\!\rangle \triangleq\ & x_1(z_1).x_1(w_1).x_1().x_2(z_2).x_2(w_2).x_2().w_1().w_2().z_1().z_2().\mathbf{0}\\ & \mid (\nu\, c_2c_2')(\overline{y_2}\langle v_2\rangle.\overline{y_2}\langle c_2\rangle.\overline{y_2}\langle\,\rangle.\mathbf{0} \mid \overline{c_2'}\langle\,\rangle.\mathbf{0})\\ & \mid (\nu\, c_1c_1')(\overline{y_1}\langle v_1\rangle.\overline{y_1}\langle c_1\rangle.\overline{y_1}\langle\,\rangle.\mathbf{0} \mid \overline{c_1'}\langle\,\rangle.\mathbf{0}) \mid \overline{k_1}\langle\,\rangle.\mathbf{0} \mid \overline{k_2}\langle\,\rangle.\mathbf{0}\end{aligned}$$

One can show that there exist Q_1, Q_2 such that $\langle\!\langle P_{79} \rangle\!\rangle \xRightarrow{\tau:x_1y_1}{}^3 \equiv Q_1 \xRightarrow{\tau:x_2y_2}{}^3 \equiv Q_2 \longrightarrow^* \equiv \mathbf{0}$.

By a similar analysis to the one for Theorem 68, we have the following result:

▶ **Theorem 84** ($\langle\!\langle \cdot \rangle\!\rangle$: Operational Correspondence). *Let $P \in \mathsf{SP}^{\mathfrak{a}}$. The following holds:*

1. *if $P \equiv (\nu\, xy)(x(w,z).P \mid \overline{y}\langle v_1, v_2\rangle \mid R)$, then $\exists Q \in \mathsf{SP}$ s.t. $\langle\!\langle P \rangle\!\rangle \longrightarrow^3 Q$ and $Q = \langle\!\langle P\{v_1/w, v_2/z\} \mid R \rangle\!\rangle$.*
2. *If $P \equiv (\nu\, xy)(x().P \mid \overline{y}\langle\,\rangle.\mathbf{0} \mid R)$, then $\exists Q \in \mathsf{SP}$ s.t. $\langle\!\langle P \rangle\!\rangle \longrightarrow Q$ and $Q = \langle\!\langle P \mid R \rangle\!\rangle$.*
3. *If $\langle\!\langle P \rangle\!\rangle \longrightarrow Q'$, then either: (i) there exists $Q \in \mathsf{SP}$ s.t. $Q' \longrightarrow^2 Q$, $P \longrightarrow P'$ and $\langle\!\langle P' \rangle\!\rangle = Q$, or (ii) $P \longrightarrow P'$ and $\langle\!\langle P' \rangle\!\rangle = Q'$.*

Using Theorem 84 and Definitions 9 and 80 we formalize our claim from Example 83: $\langle\!\langle \cdot \rangle\!\rangle$ does not add deadlocks.

▶ **Theorem 85** (Correspondence DF). *For all $P \in \mathsf{SP}^{\mathfrak{a}}$, $\mathcal{D}_{\mathbf{A}}(P)$ iff $\mathcal{D}_{\mathbf{S}}(\langle\!\langle P \rangle\!\rangle)$.*

Proof sketch. For each direction we prove its contrapositive and proceed by contradiction. See [19] for details. ◀

Using the encoding $\langle\!\langle \cdot \rangle\!\rangle$, and inspired by Theorems 84 and 85, we can define the following classes of asynchronous processes by relying on their synchronous counterparts:

▶ **Definition 86.** *The classes of asynchronous processes are defined as follows:*

$$\mathbb{C}^{\mathfrak{a}} \triangleq \{P \in \mathsf{SP}^{\mathfrak{a}} \mid \langle\!\langle P \rangle\!\rangle \in \mathbb{C}\} \quad \mathbb{P}^{\mathfrak{a}} \triangleq \{P \in \mathsf{SP}^{\mathfrak{a}} \mid \langle\!\langle P \rangle\!\rangle \in \mathbb{P}\} \quad \mu\mathbb{P}^{\mathfrak{a}} \triangleq \{P \in \mathsf{SP}^{\mathfrak{a}} \mid \langle\!\langle P \rangle\!\rangle \in \mu\mathbb{P}\}$$
$$\mathbb{H}^{\mathfrak{a}} \triangleq \{P \in \mathsf{SP}^{\mathfrak{a}} \mid \langle\!\langle P \rangle\!\rangle \in \mathbb{H}\} \quad \mathbb{H}_{\mathsf{F}}^{\mathfrak{a}} \triangleq \{P \in \mathsf{SP}^{\mathfrak{a}} \mid \langle\!\langle P \rangle\!\rangle \in \mathbb{H}_{\mathsf{F}}\} \quad \mathbb{H}_{\mathsf{R}}^{\mathfrak{a}} \triangleq \{P \in \mathsf{SP}^{\mathfrak{a}} \mid \langle\!\langle P \rangle\!\rangle \in \mathbb{H}_{\mathsf{R}}\}$$

Each asynchronous class is contained (up to $\langle\!\langle \cdot \rangle\!\rangle$) in its corresponding synchronous class:

▶ **Lemma 87.** *Let $P \in \mathsf{SP}^{\mathfrak{a}}$. The following hold:* (1) *If $P \in \mathbb{C}^{\mathfrak{a}}$, then $\langle\!\langle P \rangle\!\rangle \in \mathbb{C}$.* (2) *If $P \in \mathbb{H}^{\mathfrak{a}}$, then $\langle\!\langle P \rangle\!\rangle \in \mathbb{H}$.* (3) *If $P \in \mathbb{P}^{\mathfrak{a}}$, then $\langle\!\langle P \rangle\!\rangle \in \mathbb{P}$.* (4) *If $P \in \mu\mathbb{P}^{\mathfrak{a}}$, then $\langle\!\langle P \rangle\!\rangle \in \mu\mathbb{P}$.* (5) *If $P \in \mathbb{H}_{\mathsf{F}}^{\mathfrak{a}}$, then $\langle\!\langle P \rangle\!\rangle \in \mathbb{H}_{\mathsf{F}}$.* (6) *If $P \in \mathbb{H}_{\mathsf{R}}^{\mathfrak{a}}$, then $\langle\!\langle P \rangle\!\rangle \in \mathbb{H}_{\mathsf{R}}$.*

Proof. It follows straightforwardly from Definition 86. ◀

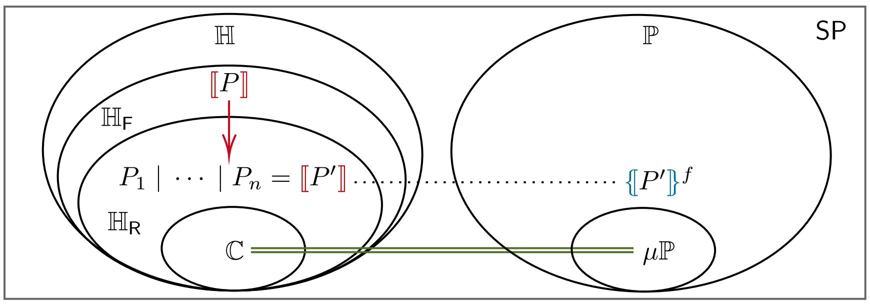
Thus, we can transfer our main results (Corollaries 53, 74, and 77) to $\mathsf{SP}^{\mathfrak{a}}$ via $\langle\!\langle \cdot \rangle\!\rangle$. For instance, Corollary 53 is transferred to $\mathsf{SP}^{\mathfrak{a}}$ as follows, using Lemma 87, and Theorem 84:

▶ **Corollary 88.** *Given $P \in \mathbb{H}_{\mathsf{F}}^{\mathfrak{a}}$, there exist $Q \in \mathbb{H}$ and $P' \in \mathbb{H}_{\mathsf{R}}^{\mathfrak{a}}$ s.t.:* (1) $[\![Q]\!] \in \textit{Dist}([\![\langle\!\langle P \rangle\!\rangle]\!])$, (2) $[\![Q]\!] \simeq^{*}_{cc} [\![\langle\!\langle P' \rangle\!\rangle]\!]$, (3) $\langle\!\langle P \rangle\!\rangle \lesssim \langle\!\langle P' \rangle\!\rangle$, (4) $[\![\langle\!\langle P \rangle\!\rangle]\!] \approx_{\mathtt{H}} [\![\langle\!\langle P' \rangle\!\rangle]\!]$.

5 Closing Remarks

Enforcing DF is a relevant issue for message-passing programs, and type systems for concurrent processes offer a convenient setting to develop analysis techniques that detect insidious circular dependencies in programs. Because there exist different type systems that enforce DF, it is natural to wonder how they compare.

We have studied this question in the general and expressive setting of the π-calculus. Our approach aims at establishing the key strengths and limitations of three representative type systems: SP, the reference language for session-typed processes; HCP, the session-typed language based on "propositions-as-sessions" and hypersequents; and P, the priority-based type system for asynchronous processes. Figure 12 gives an overview of our main technical results, refining Figure 1. We briefly discuss their broader significance. Our work can be seen as covering three dimensions of comparison: type systems (SP, HCP, P), process semantics (reduction semantics and LTS), and several definitions of DF (cf. Definitions 9, 26, 36, 60, and 80). Clearly, each type system connects these dimensions in its own way via meta-theoretical results (e.g., type preservation). From this perspective, considering a type



$\mathbb{H}$
$\mathbb{P}$
SP
$[\![P]\!]$
$\mathbb{H}_{\mathsf{F}}$
$P_1 \mid \cdots \mid P_n = [\![P']\!]$
$\{\![P']\!\}^f$
$\mathbb{H}_{\mathsf{R}}$
$\mathbb{C}$
$\mu\mathbb{P}$

Figure 12 Overview of main results (detailed version). The inclusions between the classes of processes follow from Theorem 76. The red arrow stands for the relation $\rightsquigarrow^* \simeq^*_{cc}$ (disentanglement and commuting conversions) from Corollary 53, where $P_1, \ldots, P_n \in \mathbb{C}$. This arrow, together with $\{\![P']\!\}^f \in \mathbb{P}$, represent Corollary 77. The green lines represent Corollary 74 ($\mathbb{C} = \mu\mathbb{P}$).

system such as HCP, which comes with an LTS for typed processes with very distinctive features (delayed actions and self-synchronizations), brings significant value to our results. In contrast, the work of Dardha and Pérez [9] (the most closely related work) considers different type systems for synchronous processes but only under a reduction semantics, which influences the definition(s) of DF to be considered.

Our key discovery is identifying the role that commuting conversions and disentanglement have in the class of deadlock-free processes induced by HCP (Corollary 53). Although studying the practical ramifications of our results goes beyond the scope of this paper, two observations are relevant. First, our results pave the way for developing extensions of the static analyzer TyPiCal [20] that support the mechanized analysis of (session) processes with asynchronous communication. Second, commuting conversions are already important in practical developments, such as execution strategies in *abstract machines* for session-typed programs [4] and definitions of *session subtyping* [6, 15] that enable flexible programming interfaces. An in-depth exploration of these latent connections between our results and tools for the analysis of message-passing programs is left for future work.

It is worth stressing that HCP's LTS enjoys strong logical and denotational justifications; by identifying disentanglement and commuting conversions as key notions for DF, we provide new insights into HCP and its LTS. This way, our work sheds new light into the foundations of HCP, its positioning within the line of work on "propositions-as-sessions", and its expressive power. For instance, it is surprising that processes that share more than one session are typable in HCP, in the sense of Example 14. While Kokke *et al.* [23, 22] have studied disentanglement as a tool to relate HCP and CP, here we uncover its connection with DF. Also, their work on HCP$^-$ (HCP with a reduction semantics but without delayed actions and self-synchronizations) is complementary to our findings. In this sense, our work also offers good motivations for the use of LTSs with regular transitions as the semantics for logic-based session processes (which usually rely on reduction semantics).

It is remarkable that HCP and P are not just two type systems relevant for a formal comparison, but that they are actually connected thanks to asynchrony (an important aspect not covered in Dardha and Pérez's work); the connection is made precise by Theorem 76 and Corollary 77. Here again delayed actions in HCP result to be insightful for an enhanced understanding of logic-based session types. In this line, we observe that De Young *et al.* [10] study asynchrony under "propositions-as-sessions", and in particular explain the role of commuting conversions, which in an asynchronous setting act as as structural congruences (cf. Definition 40). In the synchronous case, they correspond to behavioral equivalences [30].

An interesting point concerns the exact relation between our class $\mathbb{C}$ (Definition 34) and the class $\mathbb{L}$ in Dardha and Pérez's work. There is a key difference between the two: while they consider CP with a typing rule for independent parallel composition (cf. Rule H-Mix$_2$ in Figure 5), we only have Rule C-Mix$_0$ in Figure 4. The reason is technical: it is known that having both Mix$_0$ and Mix$_2$ induces the conflation of the units $\mathbf{1}$ and $\bot$, which conflicts with the distinction made by HCP's denotational semantics (see [19] for details). (In [9], the conflated type is denoted "$\bullet$", and there are no process constructs for session closing.) For the sake of consistency with the denotational semantics, we opted to not include this rule. Therefore, there are processes that are not in our class $\mathbb{C}$ but are in Dardha and Pérez's $\mathbb{L}$.

Having discovered the relation between DF and commuting conversions and disentanglement, an interesting item for future work is revisiting the process transformations proposed by Dardha and Pérez [9] using disentanglement. To focus on the fundamental issues of DF enforcement, we have not covered processes with recursion/sharing; we also plan to consider typed languages with constructs for shared servers and clients.

References

1 Stephanie Balzer, Bernardo Toninho, and Frank Pfenning. Manifest deadlock-freedom for shared session types. In Luís Caires, editor, *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, volume 11423 of *Lecture Notes in Computer Science*, pages 611–639. Springer, 2019. doi:10.1007/978-3-030-17184-1_22.

2 Gérard Boudol. Asynchrony and the π-calculus (note). Technical report, Rapport de Recherche 1702, INRIA, Sophia-Antipolis, 1992.

3 Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, volume 6269 of *Lecture Notes in Computer Science*, pages 222–236. Springer, 2010. doi:10.1007/978-3-642-15375-4_16.

4 Luís Caires and Bernardo Toninho. The session abstract machine. In Stephanie Weirich, editor, *Programming Languages and Systems - 33rd European Symposium on Programming, ESOP 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part I*, volume 14576 of *Lecture Notes in Computer Science*, pages 206–235. Springer, 2024. doi:10.1007/978-3-031-57262-3_9.

5 Marco Carbone and Søren Debois. A graphical approach to progress for structured communication in web services. In Simon Bliudze, Roberto Bruni, Davide Grohmann, and Alexandra Silva, editors, *Proceedings Third Interaction and Concurrency Experience: Guaranteed Interaction, ICE 2010, Amsterdam, The Netherlands, 10th of June 2010*, volume 38 of *EPTCS*, pages 13–27, 2010. doi:10.4204/EPTCS.38.4.

6 Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. On the preciseness of subtyping in session types. In Olaf Chitil, Andy King, and Olivier Danvy, editors, *Proceedings of the 16th International Symposium on Principles and Practice of Declarative Programming, Kent, Canterbury, United Kingdom, September 8-10, 2014*, pages 135–146. ACM, 2014. doi:10.1145/2643135.2643138.

7 Ornela Dardha and Simon J. Gay. A new linear logic for deadlock-free session-typed processes. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10803 of *Lecture Notes in Computer Science*, pages 91–109. Springer, 2018. doi:10.1007/978-3-319-89366-2_5.

8 Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session types revisited. In *Proc. of PPDP'12*, pages 139–150. ACM, 2012. doi:10.1145/2370776.2370794.

9 Ornela Dardha and Jorge A. Pérez. Comparing type systems for deadlock freedom. *Journal of Logical and Algebraic Methods in Programming*, 124:100717, 2022. (Preliminary version in the Proceedings of EXPRESS/SOS 2015). doi:10.1016/j.jlamp.2021.100717.

10 Henry DeYoung, Luís Caires, Frank Pfenning, and Bernardo Toninho. Cut Reduction in Linear Logic as Asynchronous Session-Typed Communication. In Patrick Cégielski and Arnaud Durand, editors, *CSL*, volume 16 of *LIPIcs*, pages 228–242. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPICS.CSL.2012.228.

11 Mariangiola Dezani-Ciancaglini, Ugo de'Liguoro, and Nobuko Yoshida. On progress for structured communications. In Gilles Barthe and Cédric Fournet, editors, *Trustworthy Global Computing, Third Symposium, TGC 2007, Sophia-Antipolis, France, November 5-6, 2007, Revised Selected Papers*, volume 4912 of *Lecture Notes in Computer Science*, pages 257–275. Springer, 2007. doi:10.1007/978-3-540-78663-4_18.

12 Simon Fowler, Wen Kokke, Ornela Dardha, Sam Lindley, and J. Garrett Morris. Separating sessions smoothly. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPIcs*, pages 36:1–36:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CONCUR.2021.36.

13 Simon Fowler, Wen Kokke, Ornela Dardha, Sam Lindley, and J. Garrett Morris. Separating sessions smoothly. *Logical Methods in Computer Science*, Volume 19, Issue 3, July 2023. doi:10.46298/lmcs-19(3:3)2023.

14 Simon J. Gay and Vasco Thudichum Vasconcelos. Linear type theory for asynchronous session types. *J. Funct. Program.*, 20(1):19–50, 2010. doi:10.1017/S0956796809990268.

15 Silvia Ghilezan, Jovanka Pantović, Ivan Prokić, Alceste Scalas, and Nobuko Yoshida. Precise subtyping for asynchronous multiparty sessions. *Proc. ACM Program. Lang.*, 5(POPL), January 2021. doi:10.1145/3434297.

16 Kohei Honda and Mario Tokoro. On asynchronous communication semantics. In *Object-Based Concurrent Computing*, number 612, in LNCS, 1992.

17 Keigo Imai, Rumyana Neykova, Nobuko Yoshida, and Shoji Yuen. Multiparty Session Programming With Global Protocol Combinators. In Robert Hirschfeld and Tobias Pape, editors, *34th European Conference on Object-Oriented Programming (ECOOP 2020)*, volume 166 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:30, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ECOOP.2020.9.

18 Jules Jacobs and Stephanie Balzer. Higher-order leak and deadlock free locks. *Proc. ACM Program. Lang.*, 7(POPL):1027–1057, 2023. doi:10.1145/3571229.

19 Juan C. Jaramillo and Jorge A. Pérez. Contrasting deadlock-free session processes (extended version), 2025. arXiv:2504.15845.

20 Naoki Kobayashi. TyPiCal: Type-based Static Analyzer for the Pi-Calculus. https://www-kb.is.s.u-tokyo.ac.jp/~koba/typical/. Accessed: 2025-03-13.

21 Naoki Kobayashi. A new type system for deadlock-free processes. In *CONCUR'06*, volume 4137 of *LNCS*, pages 233–247, 2006. doi:10.1007/11817949_16.

22 Wen Kokke, Fabrizio Montesi, and Marco Peressotti. Taking linear logic apart. In Thomas Ehrhard, Maribel Fernández, Valeria de Paiva, and Lorenzo Tortora de Falco, editors, *Proceedings Joint International Workshop on Linearity & Trends in Linear Logic and Applications, Linearity-TLLA@FLoC 2018, Oxford, UK, 7-8 July 2018*, volume 292 of *EPTCS*, pages 90–103, 2018. doi:10.4204/EPTCS.292.5.

23 Wen Kokke, Fabrizio Montesi, and Marco Peressotti. Better late than never: A fully-abstract semantics for classical processes. *Proceedings of the ACM on Programming Languages*, 3(POPL), 2019. doi:10.1145/3290337.

24 Nicolas Lagaillardie, Rumyana Neykova, and Nobuko Yoshida. Stay Safe Under Panic: Affine Rust Programming with Multiparty Session Types. In Karim Ali and Jan Vitek, editors, *36th European Conference on Object-Oriented Programming (ECOOP 2022)*, volume 222 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:29, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ECOOP.2022.4.

25 Julien Lange, Nicholas Ng, Bernardo Toninho, and Nobuko Yoshida. Fencing off go: liveness and safety for channel-based programming. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 748–761. ACM, 2017. doi:10.1145/3009837.3009847.

26 Julien Lange, Nicholas Ng, Bernardo Toninho, and Nobuko Yoshida. A static verification framework for message passing in go using behavioural types. In Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman, editors, *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pages 1137–1148. ACM, 2018. doi:10.1145/3180155.3180157.

27 Luca Padovani. Deadlock and lock freedom in the linear π-calculus. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2603088.2603116.

28 Luca Padovani. A simple library implementation of binary sessions. *J. Funct. Program.*, 27:e4, 2017. doi:10.1017/S0956796816000289.

29 Luca Padovani and Luca Novara. Types for deadlock-free higher-order programs. In Susanne Graf and Mahesh Viswanathan, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 35th IFIP WG 6.1 International Conference, FORTE 2015, Held as Part of the 10th International Federated Conference on Distributed Computing Techniques, DisCoTec 2015, Grenoble, France, June 2-4, 2015, Proceedings*, volume 9039 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2015. doi:10.1007/978-3-319-19195-9_1.

30 Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations and observational equivalences for session-based concurrency. *Information and Computation*, 239:254–302, 2014. doi:10.1016/j.ic.2014.08.001.

31 Zesen Qian, G. A. Kavvos, and Lars Birkedal. Client-server sessions in linear logic. *Proc. ACM Program. Lang.*, 5(ICFP), August 2021. doi:10.1145/3473567.

32 Davide Sangiorgi and David Walker. *PI-Calculus: A Theory of Mobile Processes*. Cambridge University Press, USA, 2001.

33 Alceste Scalas, Ornela Dardha, Raymond Hu, and Nobuko Yoshida. A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming. In Peter Müller, editor, *31st European Conference on Object-Oriented Programming (ECOOP 2017)*, volume 74 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:31, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ECOOP.2017.24.

34 Bernardo Toninho and Nobuko Yoshida. Interconnectability of session-based logical processes. *ACM Trans. Program. Lang. Syst.*, 40(4):17:1–17:42, 2018. doi:10.1145/3242173.

35 Bas van den Heuvel and Jorge A. Pérez. A gentle overview of asynchronous session-based concurrency: Deadlock freedom by typing. In Clément Aubert, Cinzia Di Giusto, Simon Fowler, and Violet Ka I Pun, editors, *Proceedings 17th Interaction and Concurrency Experience, ICE 2024, Groningen, The Netherlands, 21st June 2024*, volume 414 of *EPTCS*, pages 1–20, 2024. doi:10.4204/EPTCS.414.1.

36 Vasco T. Vasconcelos. Fundamentals of session types. *Inf. Comput.*, 217:52–70, 2012. doi:10.1016/j.ic.2012.05.002.

37 Philip Wadler. Propositions as sessions. *J. Funct. Program.*, 24(2-3):384–418, 2014. doi:10.1017/S095679681400001X.