university of groningen / faculty of science and engineering / computing science

Comparing Deadlock-Free Processes, Revisited

Channa Dias Perera
First supervisor: Prof. Dr. Jorge A. Pérez
Second supervisor: Dr. Revantha Ramanayake

September 25, 2023

Abstract

Complex concurrent systems can often release with bugs as merely detecting these bugs is difficult. An example of such a bug would be that of a concurrent process deadlocking. One method to detect deadlocks robustly is by modelling these processes and subsequently checking them via some mechanism.

A formal mode of modelling concurrent systems is through the π-calculus, processes can be checked for deadlock-freedom via a so-called type system. There are many type systems but there is very little work that formally compares them, leaving us with a limited taxonomy for organising these systems against each other.

We extend our current understanding of this taxonomy by broadening an existing comparison of two type systems. We consider a third system which is a conceptual extension of one of the type systems already compared. Notably however, this type system relies on different *operational semantics*, one which allows for non-blocking actions, which makes a direct comparison of process syntax misleading. We resolve this incongruence, leading to a set of processes relying on non-blocking actions $\mathcal{H}^\circ$, and a set of processes which do not $\mathcal{H}^\bullet$. We compare this latter set to the sets defined in the original comparison. We also provide a type-preserving translation (with examples) from $\mathcal{H}^\circ$ to $\mathcal{H}^\bullet$.

Contents

1 Introduction

Concurrent systems are more troublesome to ensure correctness for, due to a multitude of reasons. Traditional approaches to debugging usually do not work, due to the non-deterministic nature of when a process is chosen to continue its work. Novel errors, like the presence of race conditions, are present that do not exist in sequential systems. Errors may only appear well after the program has begun execution, such as a process deadlocking.

Thus, it is imperative to be able to statically check processes for such errors. One manner to do this is to model the process, and apply some automated system to check the process. One such model would be the π-calculus, whose processes can be checked for deadlock freedom, via type systems.

In this paper, we will discuss sets of processes specified in the π-calculus, first introduced by Milner et al. in [MPW92]. The π-calculus models message-passing concurrent systems, and in particular, it entertains a rich set of typing disciplines [Vas12, Yos96, IK04, TVTV13, Kob02, Kob06, CP10, Wad12, DG18], which are systems that guarantee a property for processes it deems "well-typed". However, in the papers that define these different type systems, they usually define a different syntax and semantics for the π-calculus, with respect to other papers. This makes the comparison of type systems a non-trivial task, and what is present in these papers is usually an informal comparison, by way of example.

We directly continue the work in [DP22], which compared two type systems that preserve the same property. We will conduct the same investigation as that in [DP22], except we exchange one typing system they compare, for its conceptual extension. Notably, the type systems we consider are those that guarantee that well-typed processes do not "get stuck". More formally speaking, we are interested in type systems that ensure **deadlock freedom**. This is the notion that a process, when not in its final state, will always reduce (i.e: progress to another state). Furthermore, [DP22] was interested in **session-based concurrency**, which is a form of concurrency where communication between concurrent processes can be characterised by specification of "interaction sequences", so-called **session types**. We follow them, in this regard.

[DP22] compared the type systems of [Wad12] and [Kob06], by considering the set of session-typed processes these type systems induce: $\mathcal{L}$ and $\mathcal{K}$, respectively. This comparison was worthwhile, as these type systems are following two very different lines of research. The former presents a Curry-Howard interpretation of processes, linking linear logic propositions and session types. The latter is interesting in the sense that it is quite "expressive", capturing a large number of deadlock free processes via sophisticated tracking of how channels are used. As the syntax and semantics of [Wad12] and [Kob06] differ, there is a need for a common unit of a process, in this set. This is satisfied by using processes typable under type system of [Vas12], which captures session-typed processes, including those that deadlock.

This paper will add [KMP19] to this comparison, by comparing the set of processes it induces (which we denote $\mathcal{H}$). The type system in [KMP19] is a extension of [Wad12] in the sense that they present a Curry-Howard presentation that links linear logic propositions *with hypersequents* to session types. This added comparison is valuable, as it lets us see just exactly how this extension captures more processes.

Like in [DP22], we must mention that the processes in the aforementioned sets have the property of **session progress** and but may not be deadlock-free. By session progress, we mean that these processes, when composed with other appropriate process, will progress. However, when we restrict these processes to those that are uncomposable (i.e: it does not need to interact with any outside process), such processes are deadlock free. Thus, like [DP22], we say that these sets are sets of deadlock free processes. See Definition 2.1.10 for a formal definition of (un)composable.

Now, [DP22] explored how $\mathcal{L}$ and $\mathcal{K}$ are related. In particular, they found that $\mathcal{L}$ is a strict subset of $\mathcal{K}$. Furthermore, they found a type-preserving translation of the process in $\mathcal{K}$ to those in $\mathcal{L}$. In the "Discussion" section of their paper, they suggested that if a similar analysis was done with the type system of [KMP19] (instead of [Wad12]), similar results will follow. This leads us to two of our research questions:

- *Is $\mathcal{H}$ a strict subset of $\mathcal{K}$?*
- *Is there a type-preserving translation from $\mathcal{H}$ to $\mathcal{K}$?*

Notably however, the operational semantics (which assigns meaning to the formal process syntax) in [KMP19] is non-trivially different operational semantics used by processes in [Wad12] and [Kob06]. In particular, it allows for non-blocking actions, whereas the actions in the other π-calculi are blocking. This leads us to our last research question:

- *How does one compare two type systems that follow significantly different operational semantics?*

Contributions In this paper, we define the set $\mathcal{H}$, which consists of session typed processes that have session progress (according to a variant of the type system in [KMP19]). We find that $\mathcal{H}$ is not a subset of $\mathcal{K}$, so we then split $\mathcal{H}$ into two sets: $\mathcal{H}^{\circ}$ and $\mathcal{H}^{\bullet}$. Processes in both sets have session progress with respect to the the typing system of [KMP19], but the latter has session progress with respect to the typing system of [Kob06].

We then characterise the set of processes that lie within $\mathcal{H}^{\circ}$ and show that there is a type-preserving translation from $\mathcal{H}^{\circ}$ to $\mathcal{H}^{\bullet}$. We also showed that $\mathcal{H}^{\bullet}$ and $\mathcal{L}$ are intersecting, but neither are a subset of each other. We also showed that $\mathcal{H}^{\bullet}$ is a strict subset of $\mathcal{K}$.

The characterisation of the processes in $\mathcal{H}^{\circ}$ is significant, in that it shows that processes outside $\mathcal{K}$ are those that need to leverage a non-blocking interpretation of actions. The translation is significant, in that it shows that the non-blocking actions are not essential to the process behaviour and the difference between processes in $\mathcal{H}^{\circ}$ and $\mathcal{H}^{\bullet}$ are syntactic. The separation results are significant, in that they show that the extension in [KMP19] does not make it that much more expressive (in the sense that it captures more processes with different behaviours).

Refer to Figure 1 for a visual summary of our contributions.

Paper Organisation In §2, we summarise the session π-calculus from [Vas12], and variants of the type systems present in [Wad12], [Kob06]. In §3, we present a variant of the type system present in [KMP19] and define $\mathcal{H}, \mathcal{H}^{\circ}, \mathcal{H}^{\bullet}$. In §4, we separate the sets $\mathcal{L}, \mathcal{H}^{\bullet}, \mathcal{K}$ and characterise $\mathcal{H}^{\bullet}$. In §5, we present a translation of processes in $\mathcal{H}^{\circ}$ to $\mathcal{H}^{\bullet}$. In §6, we discuss elements of future work. In §7, we discuss related work and §8 contains some concluding remarks.

Omitted technical details are included in the appendices.

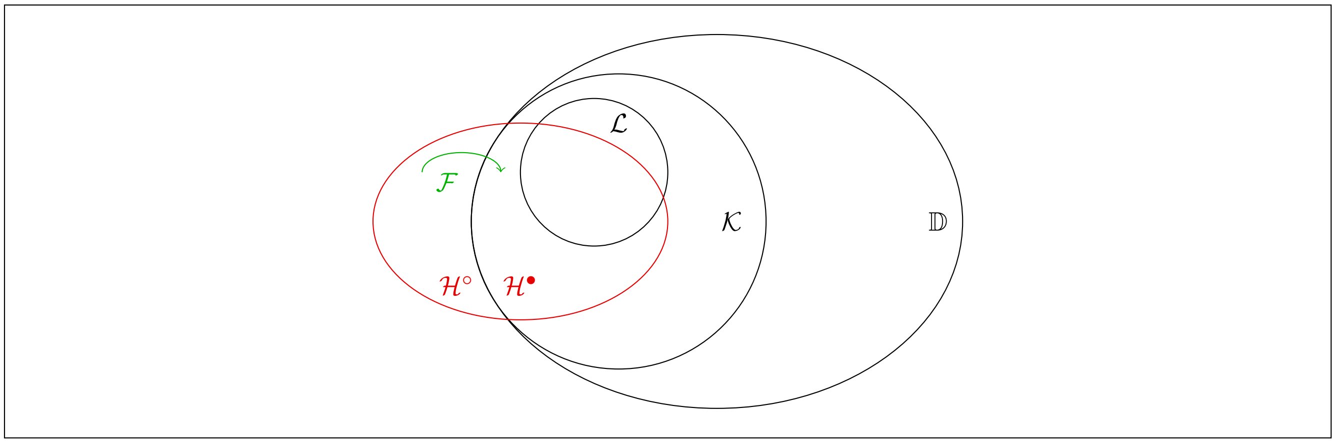

Figure 1: A summary of this paper. We use the red circle to denote the set of processes induced by the type system of [KMP19]. We use the red arrow to denote the translation of a process in $\mathcal{H}^\circ$ to a process in $\mathcal{H}^\bullet$. We use $\mathbb{D}$ to denote the set of session processes that have session progress.

2 Preliminaries

In this section, we will define the various π-calculi (and their corresponding type systems) that will be relevant in our comparison of type systems. Furthermore, in §2.4, we will introduce and apply the mechanism that allows us to resolve the differences between the π-calculi, to allow a comparison of these systems. Lastly in §2.5 we define various conventions and definitions that apply to all π-calculi.

During this section, we will also present interpretations of the formal syntax and semantics, that will help the reader understand the constructs on an intuitive level.

2.1 The Session π-calculus

This section defines the session π-calculus we will use, as defined in [DP22] which is the linear fragment of the π-calculus from [Vas12]. These processes will be the elements of the sets we define later.

We will refer to this π-calculus as the session π-calculus and we shall call processes in this calculi session processes.

Definition 2.1.1. (Session Process) Let $P, Q, \ldots$ range over session processes and $x, y, \ldots$ range over (channel) names, which are communication endpoints. Let $v, v', \ldots$ range over values.

The following BNF notation defines the syntax of our session processes:

$$\begin{aligned} P, Q &::= \overline{x}\langle v\rangle.P \;\Big|\; x(y).P \;\Big|\; x \triangleleft l_j.P \;\Big|\; x \triangleright \{l_i : P_i\}_{i\in I} \;\Big|\; \mathbf{0} \;\Big|\; P \mid Q \;\Big|\; (\boldsymbol{\nu} xy)P \\ v &::= x, y \\ \pi &::= \overline{x}\langle v\rangle \mid x(y) \mid x \triangleleft l_j \end{aligned}$$

We write $(\boldsymbol{\nu}\widetilde{xy})P$ to abbreviate $(\boldsymbol{\nu}x_1y_1)\cdots(\boldsymbol{\nu}x_ny_n)P$.

We now comment on the intended interpretation of the above syntax (given meaning by the operational semantics described later). $\overline{x}\langle v\rangle.P$ denotes the process that sends the value v along x and continues as P. $x(y).P$ denotes the process that receives input along x and continues as P, where instances of y in P acts as placeholders for the input. $x \triangleright \{l_i : P_i\}_{i \in I}$ denotes a process which continues as P_i upon receiving label l_i along x. $x \triangleleft l_j.P$ denotes a process that sends label l_j along x and continues as P. $\mathbf{0}$ denotes an inactive process and is the final state of a process. $P \mid Q$ denotes placing processes P, Q in parallel. Finally $(\nu xy)P$ denotes linking two channel endpoints in the process P.

We say that $\mid$ composes two processes. Furthermore, we refer to (νxy) as the restriction over x, y.

Definition 2.1.2. (Structural Congruence of Session Processes) We define a relation $\equiv$ between processes which, intuitively, group processes that are the same, modulo their structure. This is a **structural congruence**.

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

Remark 2.1.1. Note the scope extrusion rule for restriction: $(\nu xy)P \mid Q \equiv (\nu xy)(P \mid Q)$. This does not have a condition that x, y is free in Q, present in many other definitions. This is because we maintain Barendregt's variable convention (see §2.5) which ensures that x, y will not appear in Q.

Definition 2.1.3. (Operational Semantics for the session π-calculus) Operational semantics give meaning to the syntax presented earlier.

$$\begin{array}{lrl}
\text{(R-Com)} & (\nu xy)(\overline{x}\langle v\rangle.P \mid y(z).Q \rightarrow (\nu xy)(P \mid Q[^v/_z])) & \\
\text{(R-Case)} & (\nu xy)(x \triangleleft l_j.P \mid y \triangleright \{l_i : P_i\}_{i \in I}) \rightarrow (\nu xy)(P \mid P_j) & \text{where } j \in I \\
\text{(R-Par)} & P \rightarrow Q \implies P \mid R \rightarrow Q \mid R & \\
\text{(R-Res)} & P \rightarrow Q \implies (\nu xy)P \rightarrow (\nu xy)Q & \\
\text{(R-Str)} & P \equiv P' \wedge Q \equiv Q' \wedge P \rightarrow Q \implies P' \rightarrow Q' &
\end{array}$$

Semantics in this form are called **reduction semantics**. Generally, the form $P \rightarrow P'$ means that P will progress into P', by itself. If $P \rightarrow P'$, we say that P **reduces** to P'.

We now discuss these semantics. R-Com defines the message passing behaviour of our processes, where the value v is transmitted to Q, which it inserts into its placeholder, z, via $Q[^v/_z]$, where $[^v/_z]$ denotes capture-avoiding substitution of z as v. R-Case defines the branching communication behaviour. In both these rules, we first place a restriction between two channel endpoints, which link these two endpoints into a single channel.

R-Par defines independent reduction behaviour. R-Res defines internal communication. R-Str defines that reduction is preserved, with respect to structural equivalence.

Definition 2.1.4. (Session Types) Let $T, S, \ldots$ range over session types, given by the following BNF grammar:

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

We briefly discuss the meaning of each type. $?T.S$ is assigned to a channel that receives a value of type T and continues as type S. $!T.S$ is assigned to a channel that sends a value of type T and continues as type S. $\oplus\{l_i : S_i\}_{i \in I}$ is assigned to a channel that will continue as any of S_i if it sends l_i. $\&\{l_i : S_i\}_{i \in I}$ is assigned to a channel that will continue as any of S_i if it receives l_i. **end** is assigned to a channel that has no future actions.

Definition 2.1.5. (Session Contexts) Let $\Gamma, \Gamma', \ldots$ range over contexts, given by the following BNF grammar:

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

We say that $\emptyset$ is the **empty session context**. Session contexts are equivalent ($\equiv$) with respect to exchange (i.e: context composition is commutative) and composition with the empty context.

Intuitively, each type assigned to a name in Γ is the expected behaviour of the name n of a process that is typable under Γ.

We refer to the type T of a name n in a context Γ with the following notation: $\Gamma(n) = T$.

We say that $n : \mathbf{end}$ is a terminating type assignment (to n).

Definition 2.1.6. (Dual for Session Types)

We define the dual for session types as follows:

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

A dual of a type intuitively represents mirrored behaviour, where communication between channel endpoints of dual types can proceed safely.

Definition 2.1.7. (Unrestricted Contexts) We define the predicate un(Γ) such that

$$\text{un}(\Gamma) \iff \Gamma \equiv n_1 : \mathbf{end}, \ldots, n_l : \mathbf{end}$$

Remark 2.1.2. *Unrestricted* comes from the two types of channels in session-typed processes. There are *linear* channels, which are channels that appear in only one process, whereas *unrestricted* channels are channels that appear in (perhaps) multiple threads. Considering our focus on linear session types, the only unrestricted channels that can be typed are those that appear in zero processes (which are those channels that are finished, i.e: given a type assignment of **end**.

Definition 2.1.8. (Typing rules for the session π-calculus)

A session typing judgement is of the form $\Gamma \vdash_{\mathtt{ST}} P$ and is recursively defined. See Figure 2 for the typing rules.

We now briefly comment on the typing rules. T-Nil allows $\mathbf{0}$ to be well-typed only when all channels in its associated context are terminated. T-Par allows parallel composition of two processes, by composing their respective contexts. T-Res allows linking channel endpoints assuming these endpoints have mirrored behaviour. T-In allows input by enforcing that x only accepts inputs that have type T (and thus replacing y will that input will preserve communication fidelity). T-Out allows output of a name that has the type it intends to output. T-Brch allows branching if all the possible branching options are well-typed. T-Sel allows election if the continuation after the selection has the expected continuation.

$$\frac{\mathrm{un}(\Gamma)}{\Gamma \vdash_{\mathtt{ST}} \mathbf{0}}\ (\mathrm{T-Nil})$$

$$\frac{\Gamma_1 \vdash_{\mathtt{ST}} P \qquad \Gamma_2 \vdash_{\mathtt{ST}} Q}{\Gamma_1, \Gamma_2 \vdash_{\mathtt{ST}} P \mid Q}\ (\mathrm{T-Par}) \qquad \frac{\Gamma, x : T, y : \overline{T} \vdash_{\mathtt{ST}} P}{\Gamma \vdash_{\mathtt{ST}} (\nu xy)P}\ (\mathrm{T-Res})$$

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

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

Figure 2: Typing rules for the session π-calculus

Definition 2.1.9. (Well-Typed Session Process) We say that a session process P is well-typed if $\exists \Gamma : \Gamma \vdash_{\mathtt{ST}} P$.

We close this section with two useful definitions.

Definition 2.1.10. (Composability) If $\emptyset \vdash_{\mathtt{ST}} P$, we say that P is **uncomposable**. Otherwise, we say that P is **composable**.

We now define deadlock-freedom as defined in [DP22], which follows the definition in [KL17]:

Definition 2.1.11. (Deadlock-Freedom for Session Processes) P is **deadlock-free** if when $P \rightarrow^* P'$ where

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

then there exists an R such that $P' \rightarrow R$.

Intuitively, the idea here is that as our process reduces, if there is a request that is to be sent, it must be satisfiable and hence the process must continue to reduce.

2.2 A π-calculus based on Linear Logic

This section defines one of the π-calculus we will compare, as defined in [DP22] which is a variant of the linear fragment of the π-calculus from [Wad12].

We shall refer to this π-calculus as CP (following Wadler), and its processes CP-processes.

Definition 2.2.1. (CP-processes) Let $P, Q, \dots$ range over CP-processes and $x, y, \dots$ range over (channel) names, which are endpoints of communication channels. Let $v, v', \dots$ range over values. The following BNF grammar defines the syntax of our CP-processes:

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

We write $\overline{x}(y).P$ to abbreviate $(\nu y)\overline{x}\langle y\rangle.P$. We also write $(\nu\widetilde{x})P$ to abbreviate $(\nu x_1)\cdots(\nu x_1)P$.

The intended interpretation of this syntax follows that of the session π-calculus. The notable exception is that we have a single restriction construct $(\nu x)P$, meaning that instead of channel endpoints, we have channels. We can connect different channels (which have different names) via the link $[x \leftrightarrow y]$ and restriction operators (see the operational semantics below).

For a channel x, we represent the endpoints of this channel as: $x, \overline{x}$.

Definition 2.2.2. (Operational Semantics for CP)

(R-ChCom)	$\overline{x}\langle v\rangle \mid x(z) \to P \mid Q[{}^{v}/_{z}]$	
(R-ChCase)	$x \triangleleft l_j.P \mid x \triangleright \{l_i : P_i\}_{i\in I} \to P \mid P_j$	where $j \in I$
(R-Fwd)	$(\nu x)([x \leftrightarrow y] \mid P) \to P[{}^{y}/_{x}]$	
(R-ChRes)	$P \to Q \implies (\nu x)P \to (\nu x)Q$	

While these operational semantics look different, they capture the same essential notions as the operational semantics of the session π-calculus, after taking into account the notion that channels of the same name are already linked.

R-ChCom and R-ChCase provides meaning for message-passing / branching behaviours. Note that there is no need for a restriction operator. R-Fwd provides meaning for linking two names. R-ChRes provides meaning for internal communication.

Definition 2.2.3. (Linear Logic Types) Let $A, B, \dots$ range over linear logic propositions (without exponentiation), given by the following grammar:

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

Definition 2.2.4. (CP-Contexts) Let $\Delta, \Delta', \dots$ range over (CP-) contexts, given by the following grammar:

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

We say that $\cdot$ is the **empty CP-context**. CP-Contexts are equivalent ($\equiv$) with respect to exchange (i.e: context composition is commutative) and composition with the empty context.

We briefly describe what linear logic types mean, in a context. $x : \bot$ and $x : \mathbf{1}$ both mean that no action will take place over x. $x : A \otimes B$ describes a channel that will send a value of type A from endpoint $\overline{x}$, which will continue as type B. $x : A \parr B$ describes a channel that will receives a value of type A from endpoint x, which will continue as type B. The meanings of $x : \&\{l_i : S_i\}_{i\in I}$ and $x : \&\{l_i : S_i\}_{i\in I}$ act exactly as that for session types.

$$\frac{}{\mathbf{0} \vdash_{\text{LL}} x : \bullet}\ (\text{T-}\mathbf{1}) \qquad \frac{P \vdash_{\text{LL}} \Delta}{P \vdash_{\text{LL}} x : \bullet, \Delta}\ (\text{T-}\bot)$$

$$\frac{}{[x \leftrightarrow y] \vdash_{\text{LL}} x : A, y : \overline{A}}\ (\text{T-id}) \qquad \frac{P \vdash_{\text{LL}} \Delta, x : \overline{A} \qquad Q \vdash_{\text{LL}} \Delta', x : A}{(\nu x)\ (P \mid Q) \vdash_{\text{LL}} \Delta, \Delta'}\ (\text{T-cut}) \qquad \frac{P \vdash_{\text{LL}} \Delta \qquad Q \vdash_{\text{LL}} \Delta'}{P \mid Q \vdash_{\text{LL}} \Delta, \Delta'}\ (\text{T-mix})$$

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

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

Figure 3: Typing rules for CP

Definition 2.2.5. (Duals of Linear Logic Types)

We define the dual for linear logic types as follows:

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

The intuition for duals of linear logic propositions follow that as for session types.

In the type system that follows, we introduce a Mix rule, not present in [Wad12]. As mentioned in the discussion in [DP22], admitting such a rule means admitting $\bot = \mathbf{1}$ and so, we write $\bullet$ to denote either $\bot$ or $\mathbf{1}$, and we have $\bullet = \overline{\bullet}$.

We say that $n : \bullet$ is a terminating type assignment (to n).

Definition 2.2.6. Typing rules for CP See Figure 3 for the typing rules for CP.

We briely comment on the typing rules. T-$\mathbf{1}$ and T-$\bot$ allow type assignments (for terminated channels) in any process. T-id allows channels to be linked if they have symmetric behaviour. T-cut allows inter-process communication. However, note that no other communication actions can be amended to this channel, after composition. This is the so-called "*composition plus hiding*" principle. T-Mix allows composition of two independent processes. T-$\otimes$ allows for bound communication, meaning that we can only output values that are already "visible" to x. Furthermore, this value must be from another process (that is in parallel to it). T-$\parr$, T-$\oplus$ and T-& allow similar behaviours as T-Inp, T-Brch and T-Sel for session processes.

Definition 2.2.7. (Well-Typed CP-Processes) We say that a CP-process P is well-typed if $\exists \Delta : P \vdash_{\text{LL}} \Delta$

We now have the tools to show how CP ensures deadlock-freedom. We rephrase the theorem for deadlock-freedom in [DP22] (Theorem 3.2) as follows:

Theorem 2.2.1. *(Deadlock-Freedom for CP) If $P \vdash_{LL} \cdot$ and $P \equiv P'$ where:*

- $P' = (\nu\widetilde{n})(\overline{x}(y).Q \mid R)$
- $P' = (\nu\widetilde{n})(x(y).Q \mid R)$
- $P' = (\nu\widetilde{n})(x \lhd l_j.Q \mid R)$
- $P' = (\nu\widetilde{n})(x \rhd \{l_i : P_i\}_{i\in I} \mid R)$

then $P \to P'$, for some P'.

2.3 A π-calculus based on Channel Usage

This section defines one of the π-calculus we will compare, as defined in [DP22] which is a variant of the linear fragment of the π-calculus from [Kob06]. It is a *dyadic* π-calculus, unlike the session π-calculus and CP, which means that we can send up to two (potentially zero) values when sending a message.

We shall refer to this calculi as the dπ-calculus and its' processes dπ-processes.

Definition 2.3.1. (dπ-Processes) Let $P, Q, \ldots$ range over dπ-processes and $x, y, \ldots$ range over (channel) names, which are communication channels. Let $v, v', \ldots$ range over values. The following BNF grammar defines the syntax of our dπ processes:

$$\begin{aligned} P, Q &::= \overline{x}\langle\widetilde{v}\rangle.P \;\Big|\; x(\widetilde{z}) \;\Big|\; \textbf{case } v \textbf{ of } \{l_i\_x_i \rhd P\}_{i\in I} \;\Big|\; l_j\_v \;\Big|\; \mathbf{0} \;\Big|\; P \mid Q \;\Big|\; (\nu x)P \\ v &::= x, y \end{aligned}$$

We shall briefly comment on the intended interpretation of the above syntax. $\overline{x}\langle\widetilde{v}\rangle.P$ means that we output a 0,1,2-tuple of values along x and continue as P. $x(\widetilde{y}).P$ means that we take in as input a 0,1,2-tuple of values along x and continue as P. $\mathbf{0}$ is in the inactive process. We have the single restriction operator which binds a channel to a process, essentially establishing communication on that channel to be within itself.

Two noteworthy process constructs is the *variant value* construct l_j_v' and the **case** v **of** $\{l_i\_x_i \rhd P_i\}_{i\in I}$ construction. This construction lets us combine branching and message passing in one construction. Here, if $v = l_j\_v'$ where $j \in I$, then the process will continue as $P_j[{}^{v}/_{x_j}]$.

Like CP, channel names describe a channel (and not its endpoints). A channel x has endpoints $x, \overline{x}$.

Definition 2.3.2. (Structural Relation) We define two relations $\equiv, \preceq$ as follows:

$$P \equiv Q \iff (P \preceq Q) \wedge (Q \preceq P)$$

We define $\preceq$ as the least reflexive and transitive relation closed under the following rules:

$$\begin{aligned} P \mid Q &\equiv Q \mid P & (P \mid Q) \mid R &\equiv P \mid (Q \mid R) \\ P \mid \mathbf{0} &\equiv P & (\nu v)P \mid Q &\equiv (\nu x)(P \mid Q) \\ (\nu x)\mathbf{0} &\equiv \mathbf{0} & (\nu x)(\nu y)P &\equiv (\nu y)(\nu x)P \\ P \preceq Q \implies P \mid R &\preceq Q \mid R & P \preceq Q \implies (\nu x)P &\preceq (\nu x)Q \end{aligned}$$

$\preceq$ defines the **structural relation**, which is similar to the structural congruence except in one direction. By this we mean that $P \preceq Q$ has the meaning "P can be restructured to Q" using the rules above.

Definition 2.3.3. (Operational Semantics for the dπ-calculus)

(Rπ-Com)	$\overline{x}\langle\widetilde{v}\rangle \mid x(\widetilde{z}) \to P \mid Q[\widetilde{v}/\widetilde{z}]$
(Rπ-Case)	$\textbf{case } l_j_v \textbf{ of } \{l_i_x_i \rhd P_i\}_{i\in I} \to P_j[{}^{v}/_{x_j}]$where $j \in I$
(Rπ-Par)	$P \to Q \implies P \mid R \to Q \mid R$
(Rπ-Res)	$P \to Q \implies (\nu x)P \to (\nu x)Q$
(Rπ-Str)	$P \preceq P', P \to Q, Q' \preceq Q \implies P' \to Q'$

These rules impart meaning on dπ-processes similar how to the corresponding rules in the session π-calculus impart meaning on session processes. The only noticeably different rule is Rπ-Case, which provides the branching plus substitution behaviour discussed earlier.

Definition 2.3.4. (Deadlock-Freedom for dπ-Processes) A dπ-process P is deadlock-free if when $P \to^* P'$ where:

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

then there exists a dπ-process R such that $P' \to R$.

The intuition for deadlock-freedom here corresponds with that for deadlock-freedom for session processes.

The type system of the dπ-calculus preserves deadlock-freedom uses the notation of **usages**, which are meant to specify how a channel may be used. As well as describing the action that a channel must undergo, usages are also annotated with **capability** and **obligation** information of a certain **level**.

This information relates to an abstract global "time". An action with obligation o means that the action must be executed before time o. An action with capability c means that the action has the capability to be executed only before time c. We now formalise these notions.

Definition 2.3.5. (Channel Usages) Let $U, U', \ldots$ range over all usages. The following BNF notation defines the syntax of usages:

$$U, U' ::= 0 \;\Big|\; ?^{o}_{\kappa}.U \;\Big|\; !^{o}_{\kappa}.U \;\Big|\; U_1 \mid U_2 \;\Big|\; \uparrow^t U$$

We shall briefly comment on the meaning behind these usages. A channel with usage 0 means that the channel cannot be used at all. A channel with usage $?^{o}_{\kappa}.U$ is a channel that will be used to receive a message and then proceed to be used as U. A channel with usage $!^{o}_{\kappa}.U$ is a channel that will be used to send a message and then proceed to be used as U. A channel with usage $U_1 \mid U_2$ means that the channel can be used as U_1 and U_2 in parallel. A channel with usage $\uparrow^t U$ is a channel that is used as U, but its obligation levels are **lifted** to t.

Definition 2.3.6. (Sequential Usage / Action) We let α to denote the sequential usages (which we may sometimes call an action). If we have an action α, we say that $\overline{\alpha}$ is its co-action. Let $\alpha, \beta, \ldots$ range over all actions. The following BNF notation defines the syntax of actions:

$$\alpha, \beta ::= \,! \,|?$$

Definition 2.3.7. (Duals of actions) We define duals of actions (or co-actions), as follows:

$$\begin{aligned} \overline{!} &\triangleq ? \\ \overline{?} &\triangleq ! \end{aligned}$$

Definition 2.3.8. (Structural Relation on Usages) We will define a reduction of usages shortly, which relies on a structural relation $\preceq$ on usages, defined as follows:

$$\begin{array}{ll} U_1 \mid U_2 \preceq U_2 \mid U_1 & \uparrow^t (U_1 \mid U_2) \preceq (\uparrow^t U_1) \mid (\uparrow^t U_2) \\ (U_1 \mid U_2) \mid U_3 \preceq U_1 \mid (U_2 \mid U_3) & \\ U_1 \preceq U_1' \wedge U_2 \preceq U_2' \implies U_1 \mid U_2 \preceq U_1' \mid U_2' & \\ \uparrow^t \alpha^{\mathsf{o}}_{\kappa}.U \preceq \alpha^{\max(\mathsf{o},t)}_{\kappa}.U & U \preceq U' \implies \uparrow^t U \preceq \uparrow^t U' \end{array}$$

Intuitively, $U_1 \preceq U_2$ means that U_1 can be restructured into U_2. Observe that this definition internalises $\uparrow^t$, but never extracts it out. This is important for the reduction on usages (defined below), for it to terminate. Otherwise, the reduction can proceed indefinitely.

Definition 2.3.9. (Reduction on Usages) A reduction relation on usages is defined as follows, which conforms to intuition of usages and its structural relation.

$$\begin{array}{l} ?^{\mathsf{o}}_{\kappa}.U_1 \mid !^{\mathsf{o}'}_{\kappa'}.U_2 \to U_1 \mid U_2 \\ U \to U' \implies U \mid U'' \to U' \mid U'' \\ U \preceq U_1, U_1 \to U_2, U_2 \preceq U' \implies U \to U' \end{array}$$

This notion of reduction on usages is important, as when a process reduces, the usage that describes channels in that process must also change (i.e: reduce) in a similar manner. The rules themselves are self-explanatory.

We will now define the notion of reliability. Fundamentally, a usage is reliable if any action has a parallel co-action that has a higher capability than the action's obligation. That is to say, it can be adequately fulfilled in time.

Definition 2.3.10. (Capability / Obligation) We define two functions $\mathrm{cap}_\alpha(P), \mathrm{ob}_\alpha(P)$ as follows:

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

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

Intuitively, it captures the immediate capability / obligation information ascribed to the top level actions of a usage.

Definition 2.3.11. (Reliability) We define $\mathrm{con}_\alpha(), \mathrm{con}(), \mathrm{rel}()$ as follows:

- $\mathrm{ob}_{\overline{\alpha}}(U) \leq \mathrm{cap}_\alpha(U) \implies \mathrm{con}_\alpha(U)$
- $\mathrm{con}_?(U) \wedge \mathrm{con}_!(U) \implies \mathrm{con}(U)$
- $(\forall U' : U \to^* U' \implies \mathrm{con}(U')) \implies \mathrm{rel}(U)$

$\mathrm{con}_!(U), \mathrm{con}_?(U)$ can be thought of as asserting that a channel with usage U is has a satisfactory co-action, if it is an output / input action respectively.

$\mathrm{con}(U)$ can be thought of as asserting that a channel with usage U is reliable for the next action.

$\mathrm{rel}(U)$ can be thought of as asserting that a channel with usage U will always be reliable. Thus, if $\mathrm{rel}(U)$, we say that U is reliable.

Definition 2.3.12. (Types with Usages) Let $\tau, \tau', \ldots$ range over channel types, given by the following grammar:

$$\tau ::= \mathrm{chan}(\widetilde{\tau}; U) \mid \langle l_i : \tau_i \rangle_{i \in I}$$

Definition 2.3.13. (dπ-calculus Contexts) Let $\Gamma, \Gamma', \ldots$ range over contexts, given by the following grammar:

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

We say that $\emptyset$ is the **empty dπ-calculus context**. dπ-calculus contexts are equivalent ($\equiv$) with respect to exchange (i.e: context composition is commutative) and composition with the empty context.

If we have $x : \mathrm{chan}(\widetilde{\tau}; U)$, we mean that x is used in terms of usage U (which encodes the behaviour of the action) and sends value(s) of type $\widetilde{\tau}$.

We say that $n : \mathrm{chan}(-; \mathtt{0})$ is a terminating type assignment (to n).

When a channel is used but transfers no values, we write that the type of that channel is $\mathrm{chan}(-; U)$.

If we have $v : \langle l_i : \tau_i \rangle_{i \in I}$, we mean that a value v may be used as part of the **case of** construction.

We now repeat the auxiliary definitions in [DP22] that is used when defining the type system for the dπ-calculus.

Definition 2.3.14. (Channel recency)We define the *partial order* $\prec$, where $x \prec y$ means that channel x was created "more recently" than channel y. More precisely, if $x \prec y$, a restriction over x occurs earlier (reading the process syntax from outside in) than a restriction over y.

Definition 2.3.15. (Extending lift to types)We extend the $\uparrow^t$ operation to types:

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

Definition 2.3.16. (Extending parallel composition to types / contexts)We extend the parallel composition $\mid$ operator to types and contexts:

$$\begin{aligned}
&\mathrm{chan}(\widetilde{\tau}; U_1) \mid \mathrm{chan}(\widetilde{\tau}; U_2) \triangleq \mathrm{chan}(\widetilde{\tau}; U_1 \mid U_2) \\
&\langle l_i : \tau_i \rangle_{i \in I} \mid \langle l_i : \tau_i \rangle_{i \in I} \triangleq \langle l_i : \tau_i \rangle_{i \in I} \\
&(\Gamma_1 \mid \Gamma_2)(x) = \begin{cases} \Gamma_1(x) \mid \Gamma_2(x) & x \in \mathrm{cn}(\Gamma_1) \cap \mathrm{cn}(\Gamma_2) \\ \Gamma_1(x) & x \in \mathrm{cn}(\Gamma_1) \setminus \mathrm{cn}(\Gamma_2) \\ \Gamma_2(x) & x \in \mathrm{cn}(\Gamma_2) \setminus \mathrm{cn}(\Gamma_1) \end{cases}
\end{aligned}$$

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

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

$$\frac{}{\emptyset \vdash_\prec \mathbf{0}}(\mathrm{T}\pi\text{-Nil})$$

$$\frac{\Gamma, x:\mathrm{chan}(\widetilde\tau;U)\vdash_{\prec'} P \qquad \mathrm{rel}(U) \qquad \prec' = \prec \cup\{(x,y)\mid y\in \mathrm{fn}(P)\setminus\{x\}\}}{\Gamma\vdash_\prec (\nu x)P}(\mathrm{T}\pi\text{-Res})$$

$$\frac{\Gamma_1\vdash_\prec P_1 \qquad \Gamma_2 \vdash_\prec P_2}{\Gamma_1\mid\Gamma_2\vdash_\prec P \mid Q}(\mathrm{T}\pi\text{-Par})$$

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

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

Figure 4: The typing rules for the dπ-calculus.

Definition 2.3.17. (Type assignment compositions)We define a type assignment composition operator "$;_\prec$", which combines a type assignment to a context:

Let Γ be some context. Then $x:\mathrm{chan}(\widetilde\tau;\alpha^\circ_\kappa);_\prec \Gamma = \Gamma'$ where

$$\mathrm{cn}(\Gamma') = \triangleq \{x\}\cup \mathrm{cn}(\Gamma')$$

$$\Gamma'(x) \triangleq \begin{cases} \mathrm{chan}(\widetilde\tau;\alpha^\circ_\kappa.U) & \Gamma(x) = \mathrm{chan}(\widetilde\tau;U) \\ \mathrm{chan}(\widetilde\tau;\alpha^\circ_\kappa) & x\notin \mathrm{cn}(\Gamma) \end{cases}$$

$$\Gamma'(y) = \begin{cases} \uparrow^\kappa \Gamma(y) & y\neq x \wedge x\prec y \\ \uparrow^{\kappa+1} \Gamma(y) & y\neq x\wedge x\not\prec y \end{cases}$$

Intuitively, this operator adds the type assignment to channel x to the context Γ. If there already exists a type assignment to x, the type assignment is modified to specify that the channel will be used according to α°_κ first (assuming that the values transferred across the channel are the same).

Definition 2.3.18. (Typing system for the dπ-calculus) See Figure 4 for the typing rules for the dπ-calculus.

We now briefly comment on the typing rules. Tπ-Var, Tπ-Tup and Tπ-LVal define types for the values that can be communicated (single values, tuples of values and a labelled value).

Tπ-Nil observes that **0** can only be typed under the empty context.

Tπ-Res observes that if there is a process with a channel that has a reliable usage in P, communication along that channel can happen internally, with no outside process interacting with it. Tπ-Par observes independently parallel processes. Tπ-In, Tπ-Out gives meaning to input/output actions, by attaching an input/output action to the usage of x. Finally, Tπ-Case allows branching plus substitution if the context defines all labelled values and the substituted process expects a value of a type suggested by the label.

Definition 2.3.19. (Well-Typed dπ-Processes) We say that a dπ-process P is well-typed if $\exists\Gamma:\Gamma\vdash_\prec P$.

We have the following key result via [DP22] (Corollary 3.1):

Theorem 2.3.1. *(Deadlock-Freedom for the dπ-calculus) If $\emptyset \vdash_{\prec} P$, then P is deadlock free.*

2.4 Encodings of π-calculi

Now that we have presented the π-calculi with existing comparisons, we shall now present the relevant definitions used to perform the comparison in [DP22].

The key idea is as follows. Both CP and the dπ-calculus capture processes that follow session types. Thus, there must be some process in the session π-calculus that would map to the a process in CP or the dπ-calculus.

Thus, we define encodings that take the notions of session processes, types and typing contexts to their corresponding notions in CP and the dπ-calculus. Then, we collect the set of processes whose encodings are typable under each π-calculi and compare them.

This allows us to reconcile the syntactic differences present between these two process calculi, and compare them on common ground.

Encoding to CP We start by encoding constructions in the session π-calculus to constructs in CP.

Definition 2.4.1. (Encoding to CP-processes)

$$\begin{aligned} [\![\overline{x}\langle y\rangle.P]\!]_\ell &\triangleq \overline{x}(z).([z \leftrightarrow y] \mid [\![P]\!]_\ell) \\ [\![(\nu xy)P]\!]_\ell &\triangleq (\nu w)[\![P]\!]_\ell[{}^{w}/_{x}][{}^{w}/_{y}] \qquad \text{where } w \notin \text{fn}(P) \end{aligned}$$

$[\![\cdot]\!]_\ell$ for processes is defined as a homomorphism for other process constructs. That is to say, the outermost process construct remains verbatim and the encoding is applied to the continuation(s). For example: $[\![x(y).P]\!]_\ell = x(y).[\![P]\!]_\ell$

Encoding session processes to CP-processes involves resolving the existence of the link construct and the usage of a single restriction, as opposed to a double restriction.

We have to encode the free action, as there is no means to type the free action in the type system for CP. To do this, we link the free name to a bound name, and output that.

Encoding session types to linear logic types is straight forward:

Definition 2.4.2. (Encoding to Linear Logic types)

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

Encoding typing contexts is also straight forward:

Definition 2.4.3. (Encoding to CP-contexts)

$$\llbracket \emptyset \rrbracket_\ell \triangleq \emptyset$$
$$\llbracket \Gamma, x : T \rrbracket_\ell \triangleq \llbracket \Gamma \rrbracket_\ell, x : \llbracket T \rrbracket_\ell$$

Remark 2.4.1. [DP22] observes that the encoding is "*operationally correct*", in the sense that actions and their order are preserved, due to the direct nature of this encoding. To this end, a session process P is deadlock-free if $\llbracket P \rrbracket_\ell$ is deadlock-free.

Encoding to the dπ-calculus We now encode constructs in the session π-calculus to constructs in the the dπ-calculus, following [DP22] (which itself follows a suggestion in [Kob07] developed in [DGS12]).

First, we introduce some auxiliary notation, regarding functions, which will serve a name mappings: Let N be the set of all channel names and let $f : N \to N$. We will use the following notation for a function: $f_x = f(x)$.

Furthermore, we modify functions as follows. Let $f' = f, \{x_1, \ldots, x_n \mapsto c\}$. Then:

$$f'(n) = \begin{cases} c & n = x_i \wedge 1 \leq i \leq n \\ f(x) & \text{Otherwise} \end{cases}$$

Encoding session processes as dπ-processes is as follows:

Definition 2.4.4. (Encoding to dπ-processes)

$$\begin{aligned}
\llbracket \mathbf{0} \rrbracket^f_{\mathrm{u}} &= \mathbf{0} \\
\llbracket (\nu xy)P \rrbracket^f_{\mathrm{u}} &= (\nu c)\llbracket P \rrbracket^{f,\{x,y \mapsto c\}}_{\mathrm{u}} \\
\llbracket P \mid Q \rrbracket^f_{\mathrm{u}} &= \llbracket P \rrbracket^f_{\mathrm{u}} \mid \llbracket Q \rrbracket^f_{\mathrm{u}} \\
\llbracket \overline{x}\langle v \rangle.P \rrbracket^f_{\mathrm{u}} &= (\nu c)\overline{f_x}\langle f_v, c \rangle.\llbracket P \rrbracket^{f,\{x \mapsto c\}}_{\mathrm{u}} \\
\llbracket x(y).P \rrbracket^f_{\mathrm{u}} &= f_x(y, c).\llbracket P \rrbracket^{f,\{x, \mapsto c\}}_{\mathrm{u}} \\
\llbracket x \triangleleft l_j.P \rrbracket^f_{\mathrm{u}} &= (\nu c)\overline{f_x}\langle l_j\_c \rangle.\llbracket P \rrbracket^{f,\{x \mapsto c\}}_{\mathrm{u}} \\
\llbracket x \triangleright \{l_i : P_i\}_{i \in I} \rrbracket^f_{\mathrm{u}} &= f_x(y).\mathbf{case}\ y\ \mathbf{of}\ \{l_i\_c \triangleright \llbracket P_i \rrbracket^{f,\{x \mapsto c\}}_{\mathrm{u}}\}_{i \in I}
\end{aligned}$$

Encoding dπ-processes is straight forward for processes construct that do not involve communication directly.

For actions involving sending values: $\overline{x}\langle v \rangle.P, x \triangleleft l_j.P$, we output the value via the existing encoding, but then we continue using a new name (c), instead of the old name (x).

For actions involving receiving values: $x(y).P, x \triangleright \{l_i : P_i\}_{i \in I}$, we receive two values: the actual value (which will be received into y) but also the name that communication will continue through, since sending actions will continue via a different name (as discussed above).

Encoding session types and typing contexts is straight forward:

Definition 2.4.5. (Encoding to Types with Usages)

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

Note that when encoding the output session type $!T.S$, the second value we output is the encoded dual of S. This is because the process receiving the this value must act in a dual way to how the channel acts in the process that sent the value.

Definition 2.4.6. (Encoding to dπ-calculus contexts)

$$\begin{aligned}
[\![\emptyset]\!]^f_{\mathrm{u}} &\triangleq \emptyset \\
[\![\Gamma, x : T]\!]^f_{\mathrm{u}} &\triangleq [\![\Gamma]\!]^f_{\mathrm{u}}, f_x : [\![T]\!]_{\mathrm{u}}
\end{aligned}$$

The channel usage is worth discussing however. Observe that the usage is always simply the one action, followed by inactivity. This is because our communication actions involves switching names immediately following the communication action, which was discussed earlier.

This leads to the key result of this encoding, achieved via Proposition 3.1 and Corollary 3.2 from [DP22]:

Theorem 2.4.1. *(Encoding to the dπ-calculus preserves Deadlock-Freedom) If P is a deadlock-free session process, $[\![P]\!]^f_{\mathrm{u}}$ is a deadlock-free dπ process.*

We close this section with an important Lemma (and a corollary of it) from [DP22] (Lemma 4.4). Consider $\tau = \mathrm{chan}(\widetilde{\tau}; U)$. We write $u(\tau)$ to denote U.

Lemma 2.4.1. *(Reliability of dual session types) Let T be a session type. Then* $\mathrm{rel}(u([\![T]\!]_{\mathrm{u}}) \mid u([\![\overline{T}]\!]_{\mathrm{u}}))$.

This says that when placing dual session types in parallel, we have a reliable usage.

We have the corollary:

Corollary 2.4.1. *Let T be a session type. Then* $\mathrm{rel}(u([\![T]\!]_{\mathrm{u}} \mid [\![\overline{T}]\!]_{\mathrm{u}}))$.

Proof. Observe that $[\![T]\!]_{\mathrm{u}} = \mathrm{chan}(\widetilde{\tau}; \alpha^0_0)$ and $[\![\overline{T}]\!]_h = \mathrm{chan}(\widetilde{\tau}'; \overline{\alpha}^0_0)$. Following the proof of Lemma 4.4 in [DP22], we can show that $\widetilde{\tau}' = \widetilde{\tau}$. Thus, $u([\![T]\!]_{\mathrm{u}} \mid [\![\overline{T}]\!]_{\mathrm{u}}) = \alpha^0_0 \mid \overline{\alpha}^0_0$, which is reliable (again following the same proof in [DP22]). Thus, we have our result. ☐

2.4.1 Defining the session process sets to compare

We will now compare the sets of session processes that (once encoded) are typable under the typing systems for CP and the dπ-calculus.

First, however, we must define one auxiliary operation on typing contexts for the dπ-calculus. This is to resolve the discrepancy in typing $\mathbf{0}$, as in the dπ-calculus, only the empty set types $\mathbf{0}$, whereas in the session π-calculus, the context can be non-empty (i.e: contain terminated type assignments).

Definition 2.4.7. (Core Context) Given $\Gamma \vdash_{\texttt{ST}} P$, we write $\Gamma^{\downarrow}$ such that:

- $\Gamma = \Gamma^{\downarrow}, \Gamma'$
- $\mathrm{un}(\Gamma')$
- $\Gamma'(x) = \textbf{end} \implies x \notin \mathrm{fn}(P)$
- $\Gamma^{\downarrow}(x) = T \implies x \in \mathrm{fn}(P)$

Definition 2.4.8. (Defining sets of deadlock-free processes) Now, we can define the following sets:

$$\mathcal{L} \triangleq \{P \mid \exists \Gamma : (\Gamma \vdash_{\texttt{ST}} P \wedge [\![P]\!]_{\ell} \vdash_{\texttt{LL}} [\![\Gamma]\!]_{\ell})\}$$
$$\mathcal{K} \triangleq \{P \mid \exists \Gamma : (\Gamma \vdash_{\texttt{ST}} P \wedge [\![\Gamma^{\downarrow}]\!]^{f}_{\mathrm{u}} \vdash_{\prec} [\![P]\!]^{f}_{\mathrm{u}})\}$$

These sets contain processes that may be:

- Uncomposable: If so, they are deadlock free. For $\mathcal{L}$, this means that the encoded process is deadlock-free in the sense of Definition 2.2.1 and for $\mathcal{K}$, this means that the encoded process is deadlock-free in the sense of Definition 2.3.4.
- Composable: If so, they may deadlock. However, if composed with processes that are typable by the (currently) deadlocking processes' context, then the process will not deadlock.

One of the key results from [DP22] (Corollary 4.1) is as follows:

Theorem 2.4.2. *(Comparing $\mathcal{L}, \mathcal{K}$)*

$$\mathcal{L} \subsetneq \mathcal{K}$$

This shows that the type system of the dπ-calculus is more expressive than the type system of CP, in that it captures more deadlock-free processes.

A unification result Another key result from [DP22] is a **unification result**, in which they found two translations from $\mathcal{K}$ to $\mathcal{L}$, which is **type preserving** and maintains an **operational correspondence** (Theorem 5.1 and Theorem 5.2). The specific translations are not relevant to this thesis, but we will now take this moment to formally discuss the properties of type preservation and operational correspondence, as we will provide our own translation which should also possess these properties.

Definition 2.4.9. (Type preservation) Let $P \vdash \Gamma$, for some type system $\vdash$, for some process P and for some context Γ. A translation $[\![\cdot]\!]$ is type-preserving if:

$$[\![P]\!] \vdash \Gamma$$

This property essentially guarantees that the translated process retains **session fidelity**, the property that the process respects the protocols expected of it. In this sense, the untranslated process can be replaced with translated process without "breaking".

However, this is not sufficient, as it allows for the underlying computation to be changed. We need another property that states that this underlying computation remains. This is an **operational correspondence**. There are many formulations of operational correspondence, but we use a variant in [Gor10] (Property 3), for translation within the same process calculi:

Definition 2.4.10. (Operational Correspondence) Let P be some process, let $[\![\cdot]\!]$ be some translation of it and let $\rightarrow$ be the operational semantics for such a process. Let $\rightarrow^*$ denote the reflexive and transitive closure of $\rightarrow$. We say that $[\![\cdot]\!]$ maintains an operational correspondence if it is:

Complete : $\forall P : P \rightarrow^* Q : [\![P]\!] \rightarrow^* [\![Q]\!]$

Sound : $\forall P : [\![P]\!] \rightarrow^* Q : \exists R : P \rightarrow^* R \wedge Q \rightarrow^* [\![R]\!]$

The completeness criterion states that any sequence of transitions (i.e: computation) is retained by the translation. The second criterion states that any sequence of transitions (i.e: computation) in the translation is represented in the untranslated processes.

2.5 Conventions and Definitions For all π-calculi

This section defines some notation and conventions that all π-calculi use and follow. This includes the definition of a π-calculus that follows in §3.

Definition 2.5.1. (Prefixes) We let $\pi, \pi', \ldots$ range over prefixes, where we refer to $\overline{x}\langle v\rangle, \overline{x}(v), x(y), x \triangleleft l_j$ as prefixes.

Definition 2.5.2. (Continuation) If we have a process $P = \pi.P'$, we say that P' is its continuation. We sometimes might say that P' is the continuation of π.

Definition 2.5.3. (Subject and Object) We say that $\overline{x}\langle v\rangle$, $x(y)$, $x \triangleleft l_j$ and $x \triangleright \{l_i : P_i\}_{i\in I}$ are actions. Here, we say that x is the **subject** in each action. For the first two actions, we say v and y are **objects** in that action (respectively).

Definition 2.5.4. (Action) We denote actions as $\alpha, \beta, \ldots$ or $\pi, \rho, \phi, \ldots$. For the actions above, we write $\text{sub}(\alpha)$ to denote the subject of action α and $\text{obj}(\alpha)$ for the object of an action, if it exists. For all actions, we say that the action **occurs** or **is over** the subject.

Definition 2.5.5. (Free and bound names) We say that y is bound in $x(y).P$. We say that x, y is bound in $(\nu xy)P$. Similar notions apply to syntax that is similar. Any non-bound name that appears in a process is said to be free.

We denote the free names of a process as $\text{fn}(P)$ and the bound names of a process as $\text{bn}(P)$. We denote all names in a process $\text{cn}(P)$.

Definition 2.5.6. (Domain or channel names of a context) We denote all the names given a type in a context Γ as $\text{cn}(\Gamma)$.

Definition 2.5.7. (α-conversion) We call the process of renaming a bound variable **α-conversion** and if P and Q are the same, except for the names of bound variables, we write $P \equiv_\alpha Q$.

Definition 2.5.8. (Subprocess) Observe that all our definitions of process syntax (except for **0**) involve augmenting a process (e.g: $\overline{x}\langle v\rangle.P'$ augments P') or composing multiple processes in some way (e.g: $x \triangleright \{l_i : P_i\}_{i\in I}$). To this end, we say that the existing process (e.g: P', P_i in the processes above) is a **subprocess** of the resulting process. This definition is transitive.

Convention 2.5.1. ($\mathbf{0}$ as the continuation of a process) We omit $\mathbf{0}$ if it is the continuation of a process. That is, if we have a process $P'.\mathbf{0}$ we shall write P'.

Convention 2.5.2. (Barendregt's Variable Convention) We will follow **Barendregt's variable convention**. This means that in any mathematical context, all binding names are pairwise distinct, and are also distinct from any free name in that context. Note that this allows us to still change the names of the bound variables (as long as the convention still holds).

Convention 2.5.3. (Typing judgement proofs will omit rule names) Usually, it is obvious which rule was selected when a typing judgement proof is presented, and so, we will omit the names of the rules, for brevity's sake. This is because the syntax of process necessitates the rule that must be applied to write a specific process construct.

Proofs constructed in this way (inferring the rule used by the process syntax) is said to be constructed via an **inversion of typing judgement**.

Thus we will specify the name if it improves clarity or if the choice of rule is relevant.

3 A Hypersequent Presentation of Process Calculi

We will consider a variant of the linear linear fragment of the π-calculus presented in [KMP19], which define processes that have session types. Modifications upon the π-calculus in [KMP19] largely follow in how CP was modified in [DP22].

We shall refer to this π-calculus as HCP (following Kokke et al.) and to its processes as HCP-processes.

3.1 Hypersequent Classical Processes

Definition 3.1.1. HCP-processes Let $P, Q, \ldots$ range over HCP-processes and $x, y, \ldots$ range over (channel) names, which are communication endpoints. Let $v, v', \ldots$ range over values. The following BNF notation defines the syntax of our HCP processes:

$$\begin{aligned} P, Q &::= \overline{x}(v).P \;\Big|\; x(y).P \;\Big|\; x \triangleleft l_j.P \;\Big|\; x \triangleright \{l_i : P_i\}_{i \in I} \;\Big|\; (\nu xy)P \;\Big|\; P \mid Q \;\Big|\; \mathbf{0} \;\Big|\; [x \leftrightarrow y] \\ v &::= x, y \end{aligned}$$

Process syntax is similar to that for CP, and where the syntax is the same, the meanings are the same.

Notably, we have directly placed bound output $\overline{x}(y).P$ into our process syntax, instead of defining it through a free output and restriction. This is because we do not have a single restriction operator that allows us to do this.

Definition 3.1.2. (HCP-Context) Let $A, B, \ldots$ range over Linear Logic Types (from Definition 2.2.3). Let $\Gamma, \Gamma', \ldots$ range over contexts, given by the following grammar:

$$\Gamma, \Gamma' ::= \circ \mid \Gamma, \Gamma' \mid x : A$$

We say that $\circ$ is the **empty HCP-context**. HCP-contexts are equivalent ($\equiv$) with respect to exchange (i.e: context composition via "," is commutative) and composition with the empty context.

Definition 3.1.3. (Hypercontext) Let $\mathcal{G}, \mathcal{G}', \ldots$ range over hypercontexts, given by the following grammar:

$$\mathcal{G}, \mathcal{G}' ::= \varnothing \,\Big|\, \mathcal{G} \mid \mathcal{G}' \,\Big|\, \mathcal{G}, \Gamma$$

We say that $\varnothing$ is the **empty hypercontext**. Hypercontexts are equivalent ($\equiv$) with respect to exchange (i.e: context composition via "$|$" is commutative) and composition with the empty hypercontext.

HCP differs from CP at the typing context level. The key separating notion is that HCP encodes thread-level information at the level of types. That is to say, it encodes whether names *can* appear in parallel processes, or whether they *must* appear in the same process.

Note that (hyper)context composition (via , or $|$) is only possible if the operands being composed do not share channel names.

Note that the same notion of duals of Linear Logic Types (from Definition 2.2.5) applies in HCP as well.

Before discussing the typing system for HCP, we first discuss some auxiliary notation and definitions. We let $\bullet$ denote either $\mathbf{1}$ or $\bot$ and as such $\overline{\bullet} = \bullet$, following [DP22] in this regard. We say that $n : \bullet$ is a terminating type assignment (to n).

Definition 3.1.4. (Name Parition of a Hypercontext) We define the **name partition of a hypercontext** $\mathcal{G}$ to be the partition where a cell in the parition contains all the names in the same context. We write $\lfloor \mathcal{G} \rfloor$ to denote the name parition of $\mathcal{G}$.

Example 3.1.1. (Name Parition) Let $\mathcal{G} = x : T, y : \overline{T} \mid z : T$. Then $\lfloor \mathcal{G} \rfloor = \{\{x, y\}, \{z\}\}$

Definition 3.1.5. (Same / Different Partition) We say that $\lfloor \mathcal{G} \rfloor$ separates x, y if $\forall S \in \lfloor \mathcal{G} \rfloor : \neg(x \in S \wedge y \in S)$.

We write $x *_P y$ (respectively $x \circledast_P y$) if there is a name partition $G \in \lfloor \mathcal{G} \rfloor$ that separates (respectively does not separate) x, y, where $\mathcal{G}$ is a hypercontext such that $P \vdash_{\mathtt{H}} \mathcal{G}$.

We also write $x *_{\mathcal{G}} y$ (respectively $x \circledast_{\mathcal{G}} y$) if the name partition of $\mathcal{G}$ separates (respectively does not separate) x, y.

For convenience, if we write $x \circledast_{\cdot}\, y$ or $x *_{\cdot}\, y$, this implies that $x \neq y$.

Definition 3.1.6. (Typing system for HCP) See Figure 5 for the typing rules for HCP.

We refer to H-id, H-cut, H-Mix and H-Mix$_0$ as *structural rules* and all other rules as *logical rules.*

This is the fragment of the typing rules in [KMP19] that type linear process constructs, slightly modified to accommodate the removal of process constructs that allow communication without value transfer. This modification follows how the typing system of [Wad12] is modified in [DP22].

H-Mix$_0$ notes that $\mathbf{0}$ can be typed with the empty hypercontext. H-$\mathbf{1}$ and H-$\bot$ observe that you can annotate the terminating type assignment to any process, in any context (in the typing hypercontext). H-id observes that you can only link two channel endpoints if they have symmetric behaviors. H-Mix lets you compose any two processes in parallel (by composing their hypercontexts). H-Cut allows inter-process communication, if the two channels being restricted have symmetric behaviours. H-$\otimes$ allows bound output, if the output value is not in the same thread as the channel endpoint being used for transmission. H-$\parr$ allows input, if the placeholder for substitution is in the same thread as the channel endpoint used for transmission. H-$\oplus$ and $H - \&$ faciliate branching, in a similar manner to the type system for CP.

$$\frac{}{\mathbf{0} \vdash_{\mathtt{H}} \varnothing}\text{H-Mix}_0 \qquad \frac{P \vdash_{\mathtt{H}} \mathcal{G}}{P \vdash_{\mathtt{H}} \mathcal{G} \mid x : \bullet}\text{H-}\mathbf{1} \qquad \frac{P \vdash_{\mathtt{H}} \mathcal{G} \mid \Gamma}{P \vdash_{\mathtt{H}} \mathcal{G} \mid \Gamma, x : \bullet}\text{H-}\bot$$

$$\frac{}{[x \leftrightarrow y] \vdash_{\mathtt{H}} x : \overline{A}, y : A}\text{H-id}$$

$$\frac{P \vdash_{\mathtt{H}} \mathcal{G} \quad Q \vdash_{\mathtt{H}} \mathcal{H}}{P \mid Q \vdash_{\mathtt{H}} \mathcal{G} \mid \mathcal{H}}\text{H-Mix} \qquad \frac{P \vdash_{\mathtt{H}} \mathcal{G} \mid \Gamma, x : A \mid \Delta, y : \overline{A}}{(\nu xy)P \vdash_{\mathtt{H}} \mathcal{G} \mid \Gamma, \Delta}\text{H-Cut}$$

$$\frac{P \vdash_{\mathtt{H}} \mathcal{G} \mid \Gamma, y : A \mid \Delta, x : B}{\overline{x}(y).P \vdash_{\mathtt{H}} \mathcal{G} \mid \Gamma, \Delta, x : A \otimes B}\text{H-}\otimes \qquad \frac{P \vdash_{\mathtt{H}} \mathcal{G} \mid \Gamma, y : A, x : B}{x(y).P \vdash_{\mathtt{H}} \mathcal{G} \mid \Gamma, x : A \parr B}\text{H-}\parr$$

$$\frac{\exists j \in I : P \vdash_{\mathtt{H}} \Gamma, x : A_j}{x \lhd l_j.P \vdash_{\mathtt{H}} \Gamma, x : \oplus\{l_i : A_i\}_{i \in I}}\text{H-}\oplus \qquad \frac{\forall i \in I : P_i \vdash_{\mathtt{H}} \Gamma, x : A_i}{x \rhd \{l_i : P_i\}_{i \in I} \vdash_{\mathtt{H}} \Gamma, x : \&\{l_i : A_i\}_{i \in I}}\text{H-}\&$$

Figure 5: The typing rules for HCP

It is notable that the subprocesses P_i in the premise for H-& *must* be typable under a hypercontext that is a single context.

it is also notable that the the rules for bound output and restriction require the bound variable to be in a different context than the transmitting endpoint, similar to how in CP the bound variable is in a parallel processes than then transmitting endpoint. This conforms to the intuition that each independent context in the hypercontext contains the names that can be in two parallel component, and names in the same context must be in the same component.

Lastly, it is worth observing that given some typing judgement $P \vdash_{\mathtt{H}} \Gamma_1 \mid \cdots \mid \Gamma_n$, by repeated applications of H-$\mathbf{1}$, H-$\bot$, we can obtain:

$$P \vdash_{\mathtt{H}} \Gamma_1, \Gamma'_1 \mid \cdots \mid \Gamma_n, \Gamma'_n \mid \Gamma'_{n+1} \mid \cdots \mid \Gamma'_m$$

Where Γ_i, where $1 \leq i \leq n \leq m$, is a (possibly empty) context which only contains type assignments to $\bullet$. That is to say, we can arbitrarily add type assignments to $\bullet$ in any (perhaps new) context of the hypercontext, as long as it does not contain any names that appear in any other context.

We will utilise this in our typing judgement proofs of an HCP process. In particular, when apply H-Mix$_0$ or H-id, we will silently apply H-$\bot$, H-$\mathbf{1}$ to yield the following "axioms":

$$\frac{}{\mathbf{0} \vdash_{\mathtt{H}} \Gamma'_1 \mid \cdots \mid \Gamma'_m}\text{H-Mix}_0$$

$$\frac{}{[x \leftrightarrow y] \vdash_{\mathtt{H}} x : \overline{A}, y : A, \Gamma'_1 \mid \cdots \mid \Gamma'_m}\text{H-id}$$

Where Γ'_i is as defined earlier.

Definition 3.1.7. We say that a HCP-process P is well-typed if $\exists \mathcal{G} : P \vdash_{\mathtt{H}} \mathcal{G}$.

Before we move onto the operational semantics of HCP, an important observation that can be made is that a process is (perhaps) under typable multiple hypercontexts, in a non-trivial manner:

Example 3.1.2. (Multiple hypercontexts type the same process) Let $R = \bullet \parr \bullet, S = \bullet \otimes \bullet$

$$x(n_1).\overline{y}(n_2).\overline{z}(n_3) \vdash_{\mathtt{H}} x : R, y : S \mid z : S$$
$$x(n_1).\overline{y}(n_2).\overline{z}(n_3) \vdash_{\mathtt{H}} y : S \mid x : R \mid z : S$$

Thus, a single typing judgement can have multiple proofs (i.e: typing derivation). We see this by example:

Example 3.1.3. Multiple typing derivations for the same proof Consider

$$(\nu xy)x(n_1).\overline{y}(n_2).\overline{z}(n_3)$$

It has two typing derivations:

$$\frac{x(n_1).\overline{y}(n_2).\overline{z}(n_3) \vdash_{\mathtt{H}} x : R, y : S \mid z : S}{(\nu xz)x(n_1).\overline{y}(n_2).\overline{z}(n_3) \vdash_{\mathtt{H}} y : S}$$

and

$$\frac{x(n_1).\overline{y}(n_2).\overline{z}(n_3) \vdash_{\mathtt{H}} y : S \mid x : R \mid z : S}{(\nu xz)x(n_1).\overline{y}(n_2).\overline{z}(n_3) \vdash_{\mathtt{H}} y : S}$$

To refer to a specific proof of $P \vdash_{\mathtt{H}} \mathcal{G}$, let us define the following notation:

Notation 3.1.1. (Proofs of $P \vdash_{\mathtt{H}} \mathcal{G}$) We let $\triangledown$, $\triangledown$' range over a specific proof (derivation) of $P \vdash_{\mathtt{H}} \mathcal{G}$

Definition 3.1.8. (Subproof) Consider the proof $\triangledown$:

$$\cfrac{\cfrac{\triangledown^*}{\vdots}}{P \vdash_{\mathtt{H}} \mathcal{G}}$$

where $\triangledown^*$ is some proof. We say that $\triangledown^*$ is a **subproof** of $\triangledown$.

Given this definition of (sub)proofs, we often wish to refer to subproofs within a proof, that are rooted at the start of a named hypercontext. To this end, we define the following convention:

Convention 3.1.1. (Automatic proof naming) If we define a hypercontext $\mathcal{G}_i, \mathcal{G}', \mathcal{G}^*$ when presenting a proof $\triangledown$, we define a new proof $\triangledown_i, \triangledown', \triangledown^*$ respectively, that consists of the proof from the point of the hypercontext definition.

Example 3.1.4. (Example of automatic proof naming) If we have the proof $\triangledown$:

$$\cfrac{\cfrac{\cfrac{\vdots}{P^* \vdash_{\mathtt{H}} \mathcal{G}_1}}{P' \vdash_{\mathtt{H}} \Gamma_1 \mid \cdots \mid \Gamma_n \triangleq \mathcal{G}'}}{P \vdash_{\mathtt{H}} \mathcal{G}}$$

Then we use $\triangledown'$ to refer to

$$\cfrac{\cfrac{\vdots}{P^* \vdash_{\mathtt{H}} \mathcal{G}_1}}{P' \vdash_{\mathtt{H}} \mathcal{G}'}$$

And $\triangledown_1$ to refer to

$$\cfrac{\vdots}{P^* \vdash_{\mathtt{H}} \mathcal{G}_1}$$

And we say that $\triangledown'$ is a subproof of $\triangledown$ and $\triangledown_1$ is a subproof of both $\triangledown, \triangledown'$.

We now present the operational semantics of HCP.

Definition 3.1.9. (Action labels) Let $l, l', \ldots$ range over the set of **action labels**, given by the following grammar:

$$l ::= \overline{x}(y) \;\Big|\; x(y) \;\Big|\; x \lhd l_j \;\Big|\; x \rhd l_j \;\Big|\; [x \leftrightarrow y] \;\Big|\; \tau$$

Definition 3.1.10. (Operational semantics for HCP)

$$[x \leftrightarrow y] \xrightarrow{[x \leftrightarrow y]} \mathbf{0} \;(\mathrm{RH-Link_1}) \qquad [x \leftrightarrow y] \xrightarrow{[y \leftrightarrow x]} \mathbf{0} \;(\mathrm{RH-Link_2})$$

$$\pi.P \xrightarrow{\pi} P \;(\mathrm{RH-Action}) \qquad x \rhd \{l_i : P_i\}_{i \in I} \xrightarrow{x \rhd l_j} P_j \;(\mathrm{RH-Offer})$$

$$\frac{P \xrightarrow{l} P' \qquad \mathrm{bn}(l) \cap \mathrm{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{l} P' \mid Q} \;(\mathrm{RH-Par_1}) \qquad \frac{Q \xrightarrow{l} Q' \qquad \mathrm{bn}(l) \cap \mathrm{fn}(P) = \emptyset}{P \mid Q \xrightarrow{l} P \mid Q'} \;(\mathrm{RH-Par_2})$$

$$\frac{P \xrightarrow{[y \leftrightarrow z] P'}}{(\nu xy)P \xrightarrow{\tau} P'[x/z]} \;(\mathrm{RH-Id}) \qquad \frac{P \equiv_\alpha Q \qquad Q \xrightarrow{l} R}{P \xrightarrow{l} R} \;(\mathrm{RH-}\equiv_\alpha)$$

$$\frac{P \xrightarrow{l} P' \quad Q \xrightarrow{l'} Q' \quad \mathrm{bn}(l) \cap \mathrm{bn}(l') = \emptyset}{P \mid Q \xrightarrow{l \| l'} P' \mid Q'} \;(\mathrm{RH-Syn}) \qquad \frac{P \xrightarrow{l} \quad x, y \notin \mathrm{cn}(l) \quad x *_{P'} y}{(\nu xy)P \xrightarrow{l} (\nu xy)P'} \;(\mathrm{RH-Res})$$

$$\frac{P \xrightarrow{\overline{x}(x') \| y(y')} P'}{(\nu xy)P \xrightarrow{\tau} (\nu xy)(\nu x'y')P'} \;(\mathrm{RH-Com}) \qquad \frac{P \xrightarrow{x \lhd l_j \| x \rhd l_j} P'}{(\nu xy)P \xrightarrow{\tau} (\nu xy)P'} \;(\mathrm{RH-Case})$$

$$\frac{P \xrightarrow{l} P' \quad x \notin \mathrm{cn}(l)}{x \lhd l_j.P \xrightarrow{l} x \lhd l_j.P'} \;(\mathrm{RH-DelSel})$$

$$\frac{P \xrightarrow{l} P' \quad x, x' \notin \mathrm{cn}(l) \quad x *_{P'} x'}{\overline{x}(x').P \xrightarrow{l} \overline{x}(x').P'} \;(\mathrm{RH-DelOut}) \qquad \frac{P \xrightarrow{l} P' \quad y, y' \notin \mathrm{cn}(l) \quad y \circledast_{P'} y'}{y(y).P \xrightarrow{l} \pi.P'} \;(\mathrm{RH-DelIn})$$

$$\frac{\pi.P \xrightarrow{l} \pi.P' \quad \mathrm{fn}(\pi) *_{\pi.P} \mathrm{fn}(l)}{\pi.P \xrightarrow{\pi \| l} P'} \;(\mathrm{RH-SelfSyn})$$

This differs quite dramatically from the semantics presented earlier, which use reductions. Generally, the form of $P \xrightarrow{l} P'$ can be interpreted as "*P progresses to P' after executing action l*". Notably, we sometimes observe that $P \xrightarrow{l_1 \| l_2} P'$, which can be interpreted as "*P progresses to P' after executing actions l_1, l_2 in parallel*". Lastly, the form $P \xrightarrow{\tau} P'$ can be interpreted as "*P processes to P' without any action*", which corresponds to the notion of reduction. Such a presentation of semantics is called a **labelled transition system (LTS)**.

This is the fragment of the operational semantics in [KMP19] that type linear process constructs.

We comment on these rules briefly. RH-Link$_i$, RH-Action, RH-Offer gives semantics for progress under the core actions. RH-Par$_i$ gives semantics for independent parallel progress. RH-Id observes that if you link and then restrict channel endpoints, they form the same channel. RH-$\equiv_\alpha$ observes that actions are independent of bound name renaming. RH-Syn allows actions to occur in parallel. RH-Res allows actions to occur under restriction, if the restriction does not affect it, and the action allows x, y to stay in different parallel components. RH-Com and RH-Case allows progress evaluation without action. RH-DelSel, RH-DelOut, RH-DelIn allows delaying of selecting, output and input actions. Notably for the latter two, the action being delayed *for* must allow for correct partitioning such that the delayed action is still well-typed. Lastly, RH-SelfSyn allows for self synchronisation.

It is also worth observing that message-passing communication (via RH-Com) does not involve substitution.

Rather, the names that are transferred have a restriction placed over them. This differs from Rπ-Com, R-ChCom and R-Com.

Finally, we have the tools to state the key property of this type system. Via Corollary 3.12 in [KMP19], we have:

Theorem 3.1.1. *(Progress) If $P \not\equiv \mathbf{0}$ is well-typed, then $P \xrightarrow{l} P'$ for some l, P'.*

That is to say, if P is a well-typed process that is not in its terminating state, it can progress to some other process, under some action (delayed or otherwise).

This is quite different from the presentation of deadlock-freedom in our π-calculi, but it captures the same notion, that a process that isn't terminated will not "get stuck", that it will always progress.

3.2 Encoding HCP

Like CP and the dπ-calculus, to compare HCP to either of them, we must encode the constructs of the session π-calculus to constructs in HCP.

Definition 3.2.1. (Encoding to HCP-processes) We define $[\![\cdot]\!]_h$ on processes as follows:

$$[\![\overline{x}\langle y\rangle.P]\!]_h \triangleq \overline{x}(z).([z \leftrightarrow y] \mid [\![P]\!]_h)$$

With $[\![\cdot]\!]_h$ defined as an homomorphism for other process constructs. The reasoning is similar as for $[\![\cdot]\!]_\ell$.

Definition 3.2.2. (Encoding to Linear Logic Types / HCP-contexts) We define $[\![\cdot]\!]_h$ on types and HCP-contexts as $[\![\cdot]\!]_\ell$ (from Definitions 2.4.2, 2.4.3), however we use $[\![\cdot]\!]_h$ for clarity.

Observe we do *not* encode the session typing context into hypercontexts. This is because this thread-level information is specific to processes. Indeed, a single session typing context can result in multiple unique hypercontexts.

Thus, we define the encoding of the hypercontext once we have a process in the same mathematical context, which occurs in our definition of $\mathcal{H}$, which is the set of session typed processes that we will compare to $\mathcal{L}, \mathcal{K}$:

We define $\mathcal{H}$ as follows

Definition 3.2.3. (Processes induced by HCP)

$$\mathcal{H} \triangleq \{P \mid \exists \Gamma_1, \dots, \Gamma_k : \Gamma_1, \dots, \Gamma_k \vdash_{\mathtt{ST}} P \wedge [\![P]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_k]\!]_h\}$$

Our definition follows the same idea as in [DP22]. The notion is that we first consider the set of all session typed processes, followed by a restriction to that which is typable in HCP. Then the session typing context that would type the session process P are the session types that corresponds to the HCP contexts that type $[\![P]\!]_h$.

3.3 $\mathcal{H} \not\subseteq \mathcal{K}$

We now have the definitions and tools to begin our comparison between [KMP19] and [Kob06] (via a comparison of $\mathcal{H}$ and $\mathcal{K}$). However, consider the following session process (and its encodings):

Example 3.3.1. (Process in $\mathcal{H}$ and not in $\mathcal{K}$)

$$\begin{aligned}
P &= (\nu xy)(y(n_1).\overline{x}\langle n_2\rangle) \\
[\![P]\!]_h &= (\nu xy)(y(n_1).\overline{x}(z).[z \leftrightarrow n_2]) \\
[\![P]\!]^f_{\mathrm{u}} &= (\nu c)(c(n_1,d).(\nu e)\overline{c}\langle n_2, e\rangle)
\end{aligned}$$

$[\![P]\!]_h$ is well-typed:

$$\dfrac{\dfrac{\dfrac{[z \leftrightarrow n_2] \vdash_{\mathtt{H}} z : \bullet, n_2 : \bullet \mid x : \bullet \mid y : \bullet, n_1 : \bullet}{\overline{x}(z).[z \leftrightarrow n_2] \vdash_{\mathtt{H}} x : \bullet \otimes \bullet, n_2 : \bullet \mid y : \bullet, n_1 : \bullet}}{y(n_1).\overline{x}(z).[z \leftrightarrow n_2] \vdash_{\mathtt{H}} \bullet \otimes \bullet, n_2 : \bullet \mid y : \bullet \parr \bullet}}{[\![P]\!]_h = (\nu xy)(y(n_1).\overline{x}(z).[z \leftrightarrow n_2]) \vdash_{\mathtt{H}} n_1 : \bullet}$$

However, $[\![P]\!]^f_{\mathrm{u}}$ is not well-typed. Let $\Gamma \vdash_{\prec} c(n_1,d).(\nu e)\overline{c}\langle n_2, e\rangle$. As c is free in this process, $c \in \mathrm{cn}(\Gamma)$. Let $u(\Gamma(c)) = U$. Then, $u(\Gamma(c)) = ?^{\mathsf{o}}_{\kappa}.!^{\mathsf{o}'}_{\kappa'}$, for some $\kappa, \kappa', \mathsf{o}, \mathsf{o}'$. However, $\neg\mathrm{rel}(u(\Gamma(c)))$:

$$\begin{aligned}
\mathrm{ob}_{!}(U) = \infty > \mathsf{o} = \mathrm{cap}_{?}(U) &\implies \neg\mathrm{con}_{?}(U) \\
&\implies \neg\mathrm{con}(U) \\
&\implies \neg\mathrm{rel}(U)
\end{aligned}$$

Which leads to the following theorem:

Theorem 3.3.1.

$$\mathcal{H} \not\subseteq \mathcal{K}$$

Proof. Via Example 3.3.1. □

The reason that $P \notin \mathcal{H}$ is that it uses self synchronisation and non-blocking rules not present in the dπ-calculus to progress. Thus, comparing $\mathcal{H}$ and $\mathcal{K}$ is not fair. Rather, we should compare the processes in $\mathcal{H}$ that do not use self-synchronisation or non-blocking actions.

This motivates the following definition:

Definition 3.3.1. (Splitting $\mathcal{H}$) We define $\mathcal{H}^{\circ}, \mathcal{H}^{\bullet}$, such that $\mathcal{H} = \mathcal{H}^{\circ} \cup \mathcal{H}^{\bullet}$, where:

$$\begin{aligned}
\mathcal{H}^{\circ} &= \{P \mid P \in \mathcal{H} \land P \notin \mathcal{K}\} \\
\mathcal{H}^{\bullet} &= \{P \mid P \in \mathcal{H} \land P \in \mathcal{K}\}
\end{aligned}$$

Naturally, as $\mathcal{H}^{\bullet} \subseteq \mathcal{K}$, $P \in \mathcal{H}$ must be able to progress without self-synchronisation and non-blocking semantics. Thus, it is reasonable to compare $\mathcal{H}^{\bullet}$ to $\mathcal{L}$ and $\mathcal{K}$, as we do in the section that follows.

To observe that $\mathcal{H}^{\bullet}$ is not the empty set, see Proposition 4.1.1.

4 Comparing HCP to other Deadlock-Free π-calculi

In this section, we focus on $\mathcal{H}^{\bullet}$ and then $\mathcal{H}^{\circ}$. We will compare the sets $\mathcal{L}, \mathcal{H}^{\bullet}, \mathcal{K}$ to each other in §4.1 and then characterise the processes within $\mathcal{H}^{\circ}$, in §4.2.

4.1 Comparing $\mathcal{L}, \mathcal{H}^\bullet, \mathcal{K}$

We begin by first comparing $\mathcal{H}^\bullet$ to $\mathcal{K}$. By definition of $\mathcal{H}^\bullet$, we have that $\mathcal{H}^\bullet \subseteq \mathcal{K}$. Thus, what remains is to clarify whether $\mathcal{H}^\bullet = \mathcal{K}$.

This leads to the following Lemma:

Lemma 4.1.1.

$$\mathcal{H}^\bullet \neq \mathcal{K}$$

Proof. Consider the process P from the proof of Lemma 4.2, in [DP22] and a slightly adjusted variant of it P':

$$P = (\boldsymbol{\nu} a_1 b_1)(\boldsymbol{\nu} a_2 b_2)(a_1(x).\overline{a_2}\langle x\rangle \mid \overline{b_1}\langle n\rangle.b_2(z))$$
$$P' = (\boldsymbol{\nu} a_1 b_1)(\boldsymbol{\nu} a_2 b_2)(a_1(x).\overline{a_2}\langle x\rangle.a_1(y) \mid \overline{b_1}\langle n\rangle.b_2(z).\overline{b_1}\langle z\rangle)$$

The only change is that there is one last communication over a_1, b_1. Noting that $P \in \mathcal{K}$, it is evident that $P' \in \mathcal{K}$.

We will show that $P' \notin \mathcal{H}^\bullet$, which shows that $\mathcal{H}^\bullet \neq \mathcal{K}$.

Observe:

$$[\![P']\!]_h = (\boldsymbol{\nu} a_1 b_1)(\boldsymbol{\nu} a_2 b_2)(a_1(x).\overline{a_2}(z_1).([z_1 \leftrightarrow x] \mid a_1(y)) \mid \overline{b_1}(z_2).([n \leftrightarrow z_2] \mid b_2(z).\overline{b_1}(w).[w \leftrightarrow z]))$$

Suppose $[\![P']\!]_h \vdash_{\mathtt{H}} \mathcal{G}$ and let $\triangledown$ be a proof of this.

We define the following, for brevity:

$$P^* = (\boldsymbol{\nu} a_2 b_2)(P_1 \mid P_2)$$
$$P_1 = a_1(x).R_1 \qquad P_2 = \overline{b_1}(z_2).([n \leftrightarrow z_2] \mid Q_1)$$
$$R_1 = \overline{a_2}(z_1).R_2 \qquad Q_1 = b_2(z).Q_2$$
$$R_2 = ([z_1 \leftrightarrow x] \mid a_1(y)) \qquad Q_2 = \overline{b_1}(w).[w \leftrightarrow z]$$

Then $\triangledown$ must be:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{R_2 \vdash_{\mathtt{H}} \mathcal{R}_2}}{\overline{a_2}(z_1).R_2 \vdash_{\mathtt{H}} \mathcal{R}_1}}{a_1(x).R_1 \vdash_{\mathtt{H}} \mathcal{G}_1} \quad \dfrac{\dfrac{[n \leftrightarrow z_2] \vdash_{\mathtt{H}} n : A, z_2 : \overline{A} \quad \dfrac{\dfrac{\vdots}{Q_2 \vdash_{\mathtt{H}} \mathcal{Q}_2}}{b_2(z).Q_2 \vdash_{\mathtt{H}} \mathcal{Q}_1}}{[n \leftrightarrow z_2] \mid Q_1 \vdash_{\mathtt{H}} \mathcal{Q}'}}{\overline{b_1}(z_2).([n \leftrightarrow z_2] \mid Q_1) \vdash_{\mathtt{H}} \mathcal{G}_2}}{P_1 \mid P_2 \vdash_{\mathtt{H}} \mathcal{G}'}}{P^* \vdash_{\mathtt{H}} \mathcal{G}^*}}{[\![P']\!]_h \vdash_{\mathtt{H}} \mathcal{G}} \text{(H-Cut)}$$

We must have that $a_1 \circledast_{\mathcal{G}_1} a_2, b_1 \circledast_{\mathcal{G}_2} b_2$. This is observed as follows:

1. For P_1, observe that $a_1(x).R_1$ necessitates that $a_1 \circledast_{\mathcal{G}_1} x$ and that if $n \circledast_{\mathcal{R}_1} x$ then $a_1 \circledast_{\mathcal{G}_1} n$.

Now for $R_1 = \overline{a_2}(z_1).R_2$, we must have that if $n \circledast_{\mathcal{R}_2} z_1$ then $a_2 \circledast_{\mathcal{R}_1} n$
Observe that for $R_2 = [z_1 \leftrightarrow x] \mid a_1(y)$ we have that $z_1 \circledast_{\mathcal{R}_2} x$.
Thus we have $a_2 \circledast_{\mathcal{R}_1} x$ and therefore we have $a_1 \circledast_{\mathcal{G}_1} a_2$.

2. For $P_2 = \overline{b_1}(z_2).([n \leftrightarrow z_2] \mid Q_1)$, observe that if $b_1 \circledast_{\mathcal{Q}_1} n$, then $b_1 \circledast_{\mathcal{G}_2} n$.
   Now for $Q_1 = b_2(z).Q_2$, if $z \circledast_{\mathcal{Q}_2} n$, then $n \circledast_{\mathcal{Q}_1} b_2$.
   Then for $Q_2 = \overline{b_1}(w).[w \leftrightarrow z]$, we have that $b_1 \circledast_{\mathcal{Q}_2} z$.
   Therefore, we have $b_1 \circledast_{\mathcal{Q}_1} b_2$, meaning that $b_1 \circledast_{\mathcal{G}_2} b_2$.

Then, however we will have $a_1 \circledast_{\mathcal{G}^*} b_1$. This however would imply that you cannot apply H-Cut to P^* and our assumption that $\triangledown$ exists is false.

Thus, $P' \notin \mathcal{H}$. □

Now, we shall compare $\mathcal{H}^\bullet$ to $\mathcal{L}$. In [KMP19], a comparison was conducted between their type system and a double restriction variant of the type system by [Wad12], present in [CLM+16]. They found that well-typed process in the latter system are well-typed processes in their type system. Thus, it would be natural for $\mathcal{L} \subseteq \mathcal{H}$.

However, interestingly, this is not the case. Admitting mix principles to CP gives process syntax structures that are not found in HCP, as the following Lemma shows:

Lemma 4.1.2.

$$\mathcal{L} \nsubseteq \mathcal{H}^\bullet$$

Proof. Consider

$$P = x(y).(x(n_1) \mid y(n_2))$$

.

Then

$$[\![P]\!]_\ell = [\![P]\!]_h = x(y).(x(n_1) \mid y(n_2))$$

Let $R = \bullet \parr \bullet$.

Observe that $[\![P]\!]_\ell \vdash_{\mathtt{LL}} x : R \parr R$:

$$\dfrac{\dfrac{\dfrac{\mathbf{0} \vdash_{\mathtt{LL}} x : \bullet, n_1 : \bullet}{x(n_1) \vdash_{\mathtt{LL}} x : R} \qquad \dfrac{\mathbf{0} \vdash_{\mathtt{LL}} y : \bullet, n_2 : \bullet}{y(n_2) \vdash_{\mathtt{LL}} y : R}}{x(n_1) \mid y(n_2) \vdash_{\mathtt{LL}} x : R, y : R}}{x(y).(x(n_1) \mid y(n_2)) \vdash_{\mathtt{LL}} x : R \parr R}$$

However, $[\![P]\!]_h$ is not well-typed, which we show by contradiction. Let $\triangledown$ b a proof of $P \vdash_{\mathtt{H}} \mathcal{G}$.

Then $\triangledown$ must be:

$$\dfrac{\dfrac{\vdots}{x(n_1) \mid y(n_2) \vdash_{\mathtt{H}} \mathcal{G}'}}{x(y).(x(n_1) \mid y(n_2)) \vdash_{\mathtt{H}} \mathcal{G}} \text{ H-}\parr$$

Observe that we must have $x *_{\mathcal{G}'} y$. Thus, we cannot apply H-$\parr$ and thus, $\triangledown$ cannot exist.

Therefore, $P \notin \mathcal{H} \implies P \notin \mathcal{H}^\bullet$ and so, $\mathcal{L} \nsubseteq \mathcal{H}^\bullet$.

$\square$

Remark 4.1.1. The reason mix principles allow this result is easily seen if you consider the type system without the mix rule. In any typing rule that combined two processes in parallel, they were immediately brought into the same thread and thus, if you recreated (in [KMP19]) the same proofs of a typing judgement in [Wad12], at each level of the proof, the hypercontext would always simply be a context, and there would be no trouble applying H-$\parr$. However, admitting a mix rule does not let us guarantee that the hypercontext is a context, and which adds another item to consider when applying the HCP typing rules.

It is also possible to find a process typable in HCP but not in CP.

Lemma 4.1.3.

$$\mathcal{H}^\bullet \nsubseteq \mathcal{L}$$

Proof. Consider:

$$P = (\nu xy)(w(v).(\overline{x}\langle n_1\rangle \mid y(z) \mid w(n_2).v(n_3)))$$

Then:

$$\begin{aligned}
[\![P]\!]_\ell &= (\nu c)(w(v).(\overline{c}(z).[z \leftrightarrow n_1] \mid c(z_1) \mid w(n_2).v(n_3))) \\
[\![P]\!]_h &= (\nu xy)(w(v).(\overline{x}(z).[z \leftrightarrow n_1] \mid y(z_1) \mid w(n_2).v(n_3))) \\
[\![P]\!]_{\mathrm{u}}^f &= (\nu c)(w(v, w_1).((\nu d)\overline{c}\langle n_1, d\rangle \mid c(z, c') \mid w_1(n_2, w_2).v(n_3, v_1)))
\end{aligned}$$

We will show that $P \notin \mathcal{L} \land P \in \mathcal{H}^\bullet$, proving $\mathcal{L} \subsetneq \mathcal{H}^\bullet$.

Immediately, $[\![P]\!]_\ell$ cannot be a well-typed CP-process, since there is no parallel process under the outer restriction, which is required by T-cut. Thus $P \notin \mathcal{L}$.

Let

$$\begin{aligned}
S &= \bullet \otimes \bullet \\
T &= \bullet \parr \bullet
\end{aligned}$$

Thus, $S = \overline{T}$.

Now, observe:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[z \leftrightarrow n_1] \vdash_{\mathtt{H}} z : \bullet, n_1 : \bullet}{[z \leftrightarrow n_1] \vdash_{\mathtt{H}} z : \bullet, n_1 : \bullet \mid x : \bullet}}{\overline{x}(z).[z \leftrightarrow n_1] \vdash_{\mathtt{H}} x : S, n : \bullet} \qquad \dfrac{\mathbf{0} \vdash_{\mathtt{H}} y : \bullet, z_1 : \bullet}{y(z_1) \vdash_{\mathtt{H}} y : T} \qquad \dfrac{\dfrac{\mathbf{0} \vdash_{\mathtt{H}} v : \bullet, w : \bullet, n_2 : \bullet, n_3 : \bullet}{v(n_3) \vdash_{\mathtt{H}} v : T, w : \bullet, n_2 : \bullet}}{w(n_2).v(n_3) \vdash_{\mathtt{H}} w : T, v : T}}{\overline{x}(z).[z \leftrightarrow n_1] \mid y(z_1) \mid w(n_2).v(n_3) \vdash_{\mathtt{H}} x : S, n_1 : \bullet \mid y : T \mid w : T, v : T}}{w(v).(\overline{x}(z).[z \leftrightarrow n_1] \mid y(z_1) \mid w(n_2).v(n_3)) \vdash_{\mathtt{H}} w : T \otimes T \mid x : S, n_1 : \bullet \mid y : T}}{[\![P]\!]_h = (\nu xy)(w(v).(\overline{x}(z).[z \leftrightarrow n_1] \mid y(z_1) \mid w(n_2).v(n_3))) \vdash_{\mathtt{H}} w : T \parr T \mid n_1 : \bullet}$$

Let $E = \mathrm{chan}(-; 0), R = \mathrm{chan}(E, E; ?_0^0), S = \mathrm{chan}(E, E; !_0^0)$.

Then observe that $w : \mathrm{chan}(R, R; ?^0_0), n_1 : E \vdash_\prec P$, and so $P \in \mathcal{K}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{c}\langle n_1, d\rangle \vdash_\prec c : S, d : E, n_1 : E}{(\nu d)\overline{c}\langle n_1, d\rangle \vdash_\prec c : S, n_1 : E} \quad c(z, c') \vdash_\prec c : R \quad \dfrac{v(n_3, v_1) \vdash_\prec v : R, n_2 : E, w_2 : E,}{w_1(n_2, w_2).v(n_3, v_1) \vdash_\prec v : R, w_1 : R}}{(\nu d)\overline{c}\langle n_1, d\rangle \mid c(z, c') \mid w_1(n_2, w_2).v(n_3, v_1) \vdash_\prec c : \mathrm{chan}(E, E; !^0_0 \mid ?^0_0)), n_1 : E, v : R, w_1 : R}}{w(v, w_1).((\nu d)\overline{c}\langle n_1, d\rangle \mid c(z, c') \mid w_1(n_2, w_2).v(n_3, v_1)) \vdash_\prec c : \mathrm{chan}(E, E; !^0_0 \mid ?^0_0)), n_1 : E, v : R, w : \mathrm{chan}(R, R; ?^0_0)}}{(\nu c)(w(v, w_1).((\nu d)\overline{c}\langle n_1, d\rangle \mid c(z, c') \mid w_1(n_2, w_2).v(n_3, v_1))) \vdash_\prec n_1 : E, v : R, w : \mathrm{chan}(R, R; ?^0_0)}$$

Thus as $P \in \mathcal{K} \wedge P \in \mathcal{H}$, we see that $P \in \mathcal{H}^\bullet$.

Thus, we have that $\mathcal{L} \neq \mathcal{H}^\bullet$. $\square$

Remark 4.1.2. Now [KMP19] provided an example that is typable under HCP in [KMP19], and not in the variant of CP present in [CLM+16]. Adapting it to our syntax, it is:

$$x \triangleright \{l : P \mid Q, r : P' \mid Q'\}$$

where x appears in Q, Q'.

However, their separating process relies on T-mix being absent from the typing system in [Wad12, CLM+16], as processes in each branch ($P \mid Q$ and $P' \mid Q'$) cannot be in parallel without communication, in the type system of [Wad12, CLM+16]. As such a rule exists in the variant of CP we are discussing, we cannot use the same process as a separating process.

Proposition 4.1.1.

$$\mathcal{L} \cap \mathcal{H}^\bullet \neq \emptyset$$

Proof. Observe that $[\![\mathbf{0}]\!]_h \in \mathcal{H}$, $[\![\mathbf{0}]\!]^f_{\mathrm{u}} \in \mathcal{K}$ and so $[\![\mathbf{0}]\!]_h \in \mathcal{H}^\bullet$. Also observe that $[\![\mathbf{0}]\!]_\ell \in \mathcal{L}$. Thus $\mathbf{0} \in \mathcal{L} \cap \mathcal{H}^\bullet$ and this proves our proposition. $\square$

Collecting all these results, we have the following theorem:

Theorem 4.1.1. *(Comparing $\mathcal{L}, \mathcal{H}^\bullet, \mathcal{K}$) We have:*

1. $\mathcal{L} \cap \mathcal{H}^\bullet \neq \emptyset$
2. $\mathcal{L} \not\subseteq \mathcal{H}^\bullet \wedge \mathcal{H}^\bullet \not\subseteq \mathcal{L}$
3. $\mathcal{L} \subsetneq \mathcal{K}$
4. $\mathcal{H}^\bullet \subsetneq \mathcal{K}$

Proof. We have the first result via Proposition 4.1.1. We have the second result via Lemmas 4.1.2, 4.1.3. We have the third result via Theorem 2.4.2. We have the fourth result as follow. By definition of $\mathcal{H}^\bullet$, we have $\mathcal{H}^\bullet \subseteq \mathcal{K}$. By Lemma 4.1.1, we have $\mathcal{H}^\bullet \neq \mathcal{K}$ and so we have $\mathcal{H}^\bullet \subsetneq \mathcal{K}$. $\square$

4.2 Characterising $\mathcal{H}^\circ$

The form of processes in $\mathcal{H}^\circ$ Having compared $\mathcal{H}^\bullet$, let us now shift our focus to processes in $\mathcal{H}^o$. In particular, we aim to discover their exact structure.

To this end, it is worth recalling the proof for Theorem 3.3.1. Observe that our separation relied on a session process that encodes to a dπ-process which contains a restriction over a channel that has an unreliable usage, i.e: a usage of the form $\alpha.U'$. This corresponds to a channel which has a message passing action as the most recent action. We will show that processes that lie in $\mathcal{H}^\circ$ have at least one channel with such a usage (prior to restriction over that channel).

To illustrate this, consider a session process of the form $(\nu xy)P$. Let P have the subprocess $x(z_1).R$ and let R have the subprocess $\overline{y}\langle z_2\rangle.Q$. By Barendregt's convention, we will never have a subprocess $x(z_1).R \mid P'$ in P, where $x, y \in \mathrm{fn}(P')$.

As a consequence, the channel formed by x, y will always have a usage be of the form $\alpha.U$ and thus, that channel will always be unreliable, meaning we cannot form a restriction over it.

This is the essence of the process structures that our formal characterisation will now capture.

In our characterisation, we will need to refer to subprocesses within subprocesses, and to treat this succinctly and formally, we introduce Definitions 4.2.1 and 4.2.2.

Definition 4.2.1. (Subprocess) We define a session typed process with a hole $D[\,\cdot\,]$ as follows:

$$\begin{aligned} D[\,\cdot\,] ::= {} & \overline{x}\langle v\rangle.D'[\,\cdot\,] \\ \mid {} & x(y).D'[\,\cdot\,] \\ \mid {} & x \triangleleft l_j.D'[\,\cdot\,] \\ \mid {} & x \triangleright \{l_i : P_i\}_{i\in I} & \qquad \text{where } \exists j \in I : P_j = D'[\,\cdot\,] \\ \mid {} & D'[\,\cdot\,] \mid Q \\ \mid {} & (\nu xy)D'[\,\cdot\,] \\ \mid {} & [\,\cdot\,] \end{aligned}$$

A process with a hole is a process in which another process can be "plugged" into into, it, as the example shows:

Example 4.2.1. Consider the process with a hole: $D[\,\cdot\,] = \overline{x}\langle v\rangle.[\,\cdot\,]$. Then:

$$\begin{aligned} D[\overline{y}\langle w\rangle] &= \overline{x}\langle v\rangle.\overline{y}\langle w\rangle \\ D[z(w)] &= \overline{x}\langle v\rangle.z(w) \end{aligned}$$

Now recall H-&:

$$\frac{P_i \vdash_{\mathtt{H}} \Gamma, x : A_i \quad \forall i \in I}{x \triangleright \{l_i : P_i\}_{i\in I} \vdash_{\mathtt{H}} \Gamma, x : \&\{l_i : A_i\}_{i\in I}}\text{H-\&}$$

Observe that $x \triangleright \{l_i : P_i\}_{i\in I}$ is only typable under a context and not a hypercontext. Observe that if we have a process $(\nu xy)P'$, $P' \vdash_{\mathtt{H}} \mathcal{G}' \mid \mathcal{G}_1, x : T \mid \mathcal{G}_2 : y : \overline{T}$, meaning that P' cannot be a choice construct (i.e $x \triangleright \{l_i : P_i\}_{i\in I}$). Indeed, until an action over x or y occurs, no choice construct must be used. Thus, we define a process with a hole, without the choice construct:

Definition 4.2.2. (Subprocess without choice) We define a session typed process with a hole $C[\,\cdot\,]$ as follows:

$$\begin{aligned} C[\,\cdot\,] ::= {} & \overline{x}\langle v\rangle.C'[\,\cdot\,] \\ \mid {} & x(y).C'[\,\cdot\,] \\ \mid {} & x \triangleleft l_j.C'[\,\cdot\,] \\ \mid {} & C'[\,\cdot\,] \mid Q \\ \mid {} & (\nu xy)C'[\,\cdot\,] \\ \mid {} & [\,\cdot\,] \end{aligned}$$

Definition 4.2.3. (Non-Blocking Reliant Processes) Let P be a session process. We define the predicate $\mathrm{NBR}(\cdot)$ (read: Non-Blocking Reliant) such that

$$\mathrm{NBR}(P) \iff \exists D, C, P' : (P = D[(\nu xy)C[\pi.P']])$$

Where π is some prefix such that:

- $\mathrm{sub}(\pi) = x \vee \mathrm{sub}(\pi) = y$
- $(x = \mathrm{sub}(\pi) \wedge y \in \mathrm{fn}(P')) \oplus (y = \mathrm{sub}(\pi) \wedge x \in \mathrm{fn}(P'))$

Note that $\oplus$ is the Exclusive Or (XOR) Boolean connective. Intuitively, this definition captures all the processes that are reliant on non-blocking semantics for deadlock-freedom.

Example 4.2.2. To demonstrate this definition, let us apply it to the process in Example 3.3.1.

Observe that

$$\mathrm{NBR}((\nu xy)(y(n_1).\overline{x}\langle n_2\rangle))$$

is true, by observing that

$$(\nu xy)(y(n_1).\overline{x}\langle n_2\rangle) = D[(\nu xy)C[\pi.P']]$$

where

$$\begin{aligned} D[P^*] &= P^* \\ C[P^*] &= P^* \\ \pi &= y(n_1) \\ P' &= \overline{x}\langle n_2\rangle \end{aligned}$$

and $\mathrm{sub}(\pi) = y \wedge x \in \mathrm{fn}(P') = \{x, n_2\}$.

Utilising this definition, we now characterise $\mathcal{H}^\circ$:

Theorem 4.2.1. *(Characterising $\mathcal{H}^\circ$)*

$$\forall P \in \mathcal{H} : (P \in \mathcal{H}^\circ \iff \mathrm{NBR}(P))$$

Under the assumption $P \in \mathcal{H}$, we show:

1. $P \in \mathcal{H}^\circ \implies \mathrm{NBR}(P)$. We prove the contrapositive: $\neg\mathrm{NBR}(P) \implies P \notin \mathcal{H}^\circ$. By assuming $\neg\mathrm{NBR}(P)$, we show that $P \in \mathcal{K}$ and thus, $P \notin \mathcal{H}^\circ$.
2. $\mathrm{NBR}(P) \implies P \in \mathcal{H}^\circ$. If $\mathrm{NBR}(P)$, we show that a channel c has usage $\alpha.U'$ just before it is restricted over. As $\neg\mathrm{rel}(\alpha.U')$, we cannot form a restriction over this channel and thus, $P \notin \mathcal{K}$, leading to $P \in \mathcal{H}^\circ$.

See §*A*.1 for details.

As we close this section, we define the following useful terminology:

Definition 4.2.4. (Non-Blocking Channels) Let P be a session process. Then let:

$$\mathrm{NBC}(P) = \{(a, b) \mid \exists D, C, P' : P = D[(\nu ab)C[\pi.P']] \wedge \mathrm{sub}(\pi) \in \{a, b\}\}$$

NBC(P) (read: Non-Blocking (reliant) Channels) is the set of channels (as endpoint pairs) that rely on non-blocking semantics for communication. Any channel $(a, b) \in \mathrm{NBC}(P)$ is said to be non-blocking reliant in P. We may say (a, b) is non-blocking reliant if P is obvious from context.

Example 4.2.3. To demonstrate this definition, let us apply it to the process in Example 3.3.1

$$\mathrm{NBC}((\nu xy)y(n_1).\overline{x}\langle n_2\rangle) = \{(x, y)\}$$

5 Translating $\mathcal{H}^\circ$ to $\mathcal{H}^\bullet$

Having characterised $\mathcal{H}^\circ$, there is an interesting observation to be made.

Consider the process $P = (\nu xy)y(u).\overline{x}\langle n\rangle$. It is easy to see that NBR(P) and thus by Theorem 4.2.1, $P \in \mathcal{H}^\circ$.

Now, consider the process $P^\bullet$, which is P, except we move the action that must be non-blocking (i.e: $y(u)$ and place it in a parallel process to its continuation. That is:

$$P^\bullet = (\nu xy)(y(u) \mid \overline{x}\langle n\rangle)$$

It is plain to see that $P^\bullet$ is deadlock-free under non-blocking semantics and will be captured in the set $\mathcal{K}$.

This is the essence of a translation $\mathcal{F}.(\cdot)$ that we will define in the following section. Subsequently, we will show example of the translation in §5.2. We conclude this section with properties of the translation in §5.3.

5.1 Translating non-blocking and self-synchronizing behaviour

Key to our translation is the idea that we "split" a process into two. Take our example P from earlier. One parallel process had actions related to x and another had actions related to y.

Another perspective on this is one parallel process has actions that *keep* actions related to x and another *removes* actions related to y.

We formalise these notions as $\mathcal{K}(\cdot)$ and $\mathcal{R}(\cdot)$ respectively.

We demonstrate them, by way of informal example:

Example 5.1.1. Now

$$\mathcal{R}^{\{x,n\}}(y(u).\overline{x}\langle n\rangle)$$

is the process after the removal of actions related to x, n from the process. This will result in

$$y(u).\mathcal{R}^{\{x,n\}}(\overline{x}\langle n\rangle)$$

As $y(u)$ should not be removed from the result. Continuing on, this results in:

$$y(u).\mathbf{0}$$

as $\overline{x}\langle n\rangle$ is removed.

Similarly,

$$\mathcal{K}^{\{x,n\}}(y(u).\overline{x}\langle n\rangle)$$

is the process after keeping actions *only* related to x, n from the process. This results in

$$\mathcal{K}^{\{x,n\}}(\overline{x}\langle n\rangle)$$

As no name in $y(u)$ is in $\{x, n\}$. Continuing on, this results in:

$$\overline{x}\langle n\rangle$$

As $\overline{x}\langle n\rangle$ contains names in $\{x, n\}$.

Placing these in parallel

$$\mathcal{R}^{\{x,n\}}(y(u).\overline{x}\langle n\rangle) \mid \mathcal{K}^{\{x,n\}}(y(u).\overline{x}\langle n\rangle)$$

we will have:

$$y(u) \mid \overline{x}\langle n\rangle$$

as expected.

Remark 5.1.1. $\mathcal{R}(P), \mathcal{K}(P)$ both require information from the hypercontext that types P. However both $\mathcal{R}(P)$ and $\mathcal{K}(P)$ are recursive upon subprocesses in P. Thus if our procedure was parameterised by a hypercontext that types P, it will not be well-defined as recursive applications of the procedure will have a choice of hypercontexts that type the subprocess it is being applied to.

However, it is well-defined upon a *proof* of a hypercontext, as the recursive calls have one hypercontext to refer to: the hypercontext that types the subprocess in the proof.

We now formalise these intermediate procedures as follows:

Definition 5.1.1. (*Keeping* constructs acting on certain names in a process) Given a well-typed session process P, a set of names S and a proof $\triangledown$ of $[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}$, we define the session process $\mathcal{K}^S_\triangledown(P)$ as follows:

$$\mathcal{K}^S_\triangledown(P) = \begin{cases} \overline{x}\langle v\rangle.\mathcal{K}^{S\setminus\{v\}}_{\triangledown'}(R) & P = \overline{x}\langle y\rangle.R \land x \in S \\ x(y).\mathcal{K}^{S\cup\{y\}}_{\triangledown'}(R) & P = x(y) \land x \in S \\ x \triangleleft l_j.\mathcal{K}^{S}_{\triangledown'}(R) & P = x \triangleleft l_j.R \land x \in S \\ \mathcal{K}^{S}_{\triangledown'}(R) & P = \pi.R \land \mathrm{sub}(\pi) \notin S \\ x \triangleright \{l_i : P_i\}_{i\in I} & x \triangleright \{l_i : P_i\}_{i\in I} \land (x \in S \lor \exists i : \exists n \in S : n \in \mathrm{fn}(P_i)) \\ \mathcal{K}^{S\setminus\mathrm{fn}(P_2)}_{\triangledown_1}(P_1) \mid \mathcal{K}^{S\setminus\mathrm{fn}(P_1)}_{\triangledown_2}(P_2) & P = P_1 \mid P_2 \\ (\boldsymbol{\nu} xy)(\mathcal{K}^{S\cup\{x,y\}}_{\triangledown'}(R)) & P = (\boldsymbol{\nu} xy)R \land \exists n \in S : n \in \mathrm{cn}(\Delta_1) \cup \mathrm{cn}(\Delta_2) \\ \mathcal{K}^{S}_{\triangledown'}(R) & P = (\boldsymbol{\nu} xy)R \land \neg(\exists n \in S : n \in \mathrm{cn}(\Delta_1) \cup \mathrm{cn}(\Delta_2)) \\ \mathbf{0} & \text{Otherwise} \end{cases}$$

Where $\Delta_i, \triangledown', \triangledown_i$ are defined as follows (recalling Convention 3.1.1):

When $P = (\nu xy)R$, $\triangledown$ must be:

$$\dfrac{\dfrac{\vdots}{[\![R]\!]_h \vdash_{\mathtt{H}} \mathcal{G}' \mid \Delta_1, x : T \mid \Delta_2, y : \overline{T} \triangleq \triangledown'}}{[\![(\nu xy)R]\!]_h \vdash_{\mathtt{H}} \mathcal{G}' \mid \Delta_1, \Delta_2}$$

When $P = P_1 \mid P_2$, $\triangledown$ must be:

$$\dfrac{\triangledown_1 \qquad \triangledown_2}{[\![P_1 \mid P_2]\!]_h \vdash_{\mathtt{H}} \mathcal{R}_1 \mid \mathcal{R}_2}$$

where $\triangledown_i$ is a proof of $P_i \vdash_{\mathtt{H}} \mathcal{R}_i$.

Otherwise, $\triangledown$ must be:

$$\dfrac{\triangledown'}{[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}}$$

We briefly comment on each case (and similar but dual comments apply to the next intermediate definition, Definition 5.1.2). The first three cases are to do with prefix actions which contain a subject which we wish to keep in the result. In all these cases, the prefix is present in the resulting process, with the continuation being transformed. Where these transformation differs is in the names it continues to keep, augmented based on the object of the action. For the first case, we do not keep the free name v, as we do not need to bring it with x. For the second case, we must keep y in the same process as x, as we must substitute y after the input action. For the last case, there is no object, and thus we do not have to modify S.

The fourth case involves a prefix we do not keep, and the resulting process does not include that prefix. The last action construct we consider is the choice construct. In this case, we keep the choice construct if it contains a name that we wish to keep (either as a subprocess or as the subject in the action). The parallel composition rule applies the procedure recursively, making sure to remove names that we wish to keep if we know it cannot be in that process. The rules to do with the restriction operator include the restriction if at least the restricted endpoint must appear in the same context as the names we are keeping in the same process (information we can obtain via the hypercontext $\mathcal{G}' \mid \Delta_1, x : T \mid \Delta_2, y : \overline{T}$).

Definition 5.1.2. (*Removing* constructs acting on certain names from a process) Given a well-typed session process P, a set of names S and a proof $\triangledown$ of $[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}$, we define the session process $\mathcal{R}^S_\triangledown(P)$ as follows:

$$\mathcal{R}^S_{\mathcal{G}}(P) = \begin{cases} \mathcal{R}^{S\setminus\{v\}}_{\mathcal{G}'}(R) & P = \overline{x}\langle v\rangle.R \wedge x \in S \\ \mathcal{R}^{S\cup\{y\}}_{\mathcal{G}'}(R) & P = x(y).R \wedge x \in S \\ \mathcal{R}^{S}_{\mathcal{G}'}(R) & P = x \triangleleft l_j.R \wedge x \in S \\ \pi.\mathcal{R}^{S}_{\mathcal{G}'}(R) & P = \pi.R \wedge \mathrm{sub}(\pi) \notin S \\ \mathbf{0} & x \triangleright \{l_i : P_i\}_{i\in I} \wedge (x \in S \vee \exists i : \exists n \in S : n \in \mathrm{fn}(P_i)) \\ \mathcal{R}^{S\setminus\mathrm{fn}(P_2)}_{\mathcal{G}_1}(P_1) \mid \mathcal{R}^{S\setminus\mathrm{fn}(P_1)}_{\mathcal{G}_2}(P_2) & P = P_1 \mid P_2 \\ (\nu xy)(\mathcal{R}^{S}_{\mathcal{G}'}(R)) & P = (\nu xy)R \wedge \neg(\exists n \in S : n \in \mathrm{cn}(\Delta_1) \cup \mathrm{cn}(\Delta_2)) \\ \mathcal{R}^{S\cup\{x,y\}}_{\mathcal{G}'}(R) & P = (\nu xy)R \wedge \exists n \in S : n \in \mathrm{cn}(\Delta_1) \cup \mathrm{cn}(\Delta_2) \\ P & \text{Otherwise} \end{cases}$$

Where $\Delta_i, \triangledown', \triangledown_i$ are defined as follows (recalling Convention 3.1.1):

When $P = (\nu xy)R$, $\triangledown$ must be:

$$\cfrac{\cfrac{\vdots}{[\![R]\!]_h \vdash_{\mathtt{H}} \mathcal{G}' \mid \Delta_1, x : T \mid \Delta_2, y : \overline{T} \triangleq \triangledown'}}{[\![(\nu xy)R]\!]_h \vdash_{\mathtt{H}} \mathcal{G}' \mid \Delta_1, \Delta_2}$$

When $P = P_1 \mid P_2$, $\triangledown$ must be:

$$\frac{\triangledown_1 \qquad \triangledown_2}{[\![P_1 \mid P_2]\!]_h \vdash_{\mathtt{H}} \mathcal{R}_1 \mid \mathcal{R}_2}$$

where $\triangledown_i$ is a proof of $P_i \vdash_{\mathtt{H}} \mathcal{R}_i$.

Otherwise, $\triangledown$ must be:

$$\frac{\triangledown'}{[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}}$$

Now having defined our auxiliary functions, we can resolve one channel by following our aforementioned approach. We define the sets N_1, N_2 to move all the names in the same context as x, y, as they must remain in the same process.

Definition 5.1.3. (Parallelizing *one* non-blocking reliant channel) Given a well-typed session process P and a proof $\triangledown$ of $[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}$, we define the set of session process $\mathcal{F}'_{\triangledown}(P)$ as follows:

$$\mathcal{F}'_{\triangledown}(P) = \begin{cases} D[(\nu xy)(\mathcal{R}^{N_1}_{\triangledown^*}(C[\pi.P']) \mid \mathcal{K}^{N_1}_{\triangledown^*}(C[\pi.P'])] & P = D[(\nu xy)C[\pi.P']] \wedge x = \mathrm{sub}(\pi) \wedge y \in \mathrm{fn}(P') \\ D[(\nu xy)(\mathcal{R}^{N_2}_{\triangledown^*}(C[\pi.P']) \mid \mathcal{K}^{N_2}_{\triangledown^*}(C[\pi.P']))] & P = D[(\nu xy)C[\pi.P']] \wedge y = \mathrm{sub}(\pi) \wedge x \in \mathrm{fn}(P') \\ P & \text{Otherwise} \end{cases}$$

We must have $\triangledown$ be:

$$\cfrac{\cfrac{\cfrac{\cfrac{\vdots}{[\![C[\pi.P']]\!]_h \vdash_{\mathtt{H}} \mathcal{G}'' \mid \Delta, x : T \mid \Theta, y : \overline{T} \triangleq \triangledown^*}}{[\![(\nu xy)C[\pi.P']]\!]_h \vdash_{\mathtt{H}} \mathcal{G}'}}{\vdots}}{[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}}$$

Then, we have the following definitions:

$$N_1 = \{x\} \cup \mathrm{cn}(\Delta)$$
$$N_2 = \{y\} \cup \mathrm{cn}(\Theta)$$

Finally, we apply $\mathcal{F}'_{\cdot}(\cdot)$ repeatedly, to resolve all channels, as follows:

Definition 5.1.4. (Parallelizing *all* non-blocking reliant channels) Given a well-typed session process P and a proof $\triangledown$ of $[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}$, we define the set of session process $\mathcal{R}^S_{\triangledown}(P)$ as follows:

$$\mathcal{F}_{\triangledown}(P) = \begin{cases} \bigcup_{\triangledown'} \mathcal{F}_{\triangledown'}(\mathcal{F}'_{\triangledown}(P)) & \mathrm{NBR}(P) \\ \{P\} & \text{Otherwise} \end{cases}$$

where $\triangledown'$ is a proof of $\mathcal{F}'_{\triangledown}(P) \vdash_{\mathtt{H}} \mathcal{G}$

5.2 Examples of the translation

Before we define properties upon this translation, let us first apply this translation to some examples, to both provide a better understanding of its mechanics, as well as to motivate the properties that follow.

Furthermore, to motivate an operational correspondence conjecture we will present, we will enumerate the transitions of the example process and the translated processes, and compare them.

Example 5.2.1. (Simple translation) This example shows a simple translation, taking a process that requires on non-blocking semantics to be deadlock-free to one that is deadlock-free under blocking semantics.

Consider

$$P = (\nu xy)(y(n_1).\overline{x}\langle n_2\rangle)$$

And the following proof $\triangledown$ of $[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{[z \leftrightarrow n_2] \vdash_{\mathtt{H}} z : \bullet, n_2 : \bullet}{[z \leftrightarrow n_2] \vdash_{\mathtt{H}} z : \bullet, n_2 : \bullet \mid x : \bullet \mid y : \bullet, n_1 : \bullet}}{\overline{x}(z).[z \leftrightarrow n_2] \vdash_{\mathtt{H}} n_2 : \bullet, x : \bullet \otimes \bullet \mid y : \bullet, n_1 : \bullet \triangleq \mathcal{G}_2}}{y(n_1).\overline{x}(z).[z \leftrightarrow n_2] \vdash_{\mathtt{H}} n_2 : \bullet, x : \bullet \otimes \bullet \mid y : \bullet \parr \bullet \triangleq \mathcal{G}_1}}{[\![P]\!]_h = (\nu xy)(y(n_1).\overline{x}(z).[z \leftrightarrow n_2]) \vdash_{\mathtt{H}} n_2 : \bullet \triangleq \mathcal{G}}$$

Then:

$$\begin{aligned} \mathcal{F}'_{\triangledown}(P) &= (\nu xy)(\mathcal{R}^{\{n_2,x\}}_{\triangledown_1}(y(n_1).\overline{x}\langle n_2\rangle) \mid \mathcal{K}^{\{n_2,x\}}_{\triangledown_1}(y(n_1).\overline{x}\langle n_2\rangle)) \\ &= (\nu xy)(y(n_1).\mathcal{R}^{n_2,x}_{\triangledown_2}(\overline{x}\langle n_2\rangle) \mid \mathcal{K}^{n_2,x}_{\triangledown_2}(\overline{x}\langle n_2\rangle)) \\ &= (\nu xy)(y(n_1) \mid \overline{x}\langle n_2\rangle) \end{aligned}$$

As we can see, this translation correctly splits the process, in an expected manner. It is instructive to observe that the encoding of the free output $\overline{x}\langle n_2\rangle$ poses no trouble to the translation here. We will see a more representative example of this in the following example.

Furthermore, we can observe that the translation was type preserving on this example. Let $R = \bullet \parr \bullet, S = \bullet \otimes \bullet$:

$$\dfrac{\dfrac{\dfrac{\mathbf{0} \vdash_{\mathtt{H}} y : \bullet, n_1 : \bullet}{y(n_1) \vdash_{\mathtt{H}} y : R} \quad \dfrac{[z \leftrightarrow n_2] \vdash_{\mathtt{H}} z : \bullet, n_2 : \bullet \mid x : \bullet}{\overline{x}(z).[z \leftrightarrow n_2] \vdash_{\mathtt{H}} x : S, n_2 : \bullet}}{y(n_1) \mid \overline{x}(z).[z \leftrightarrow n_2] y : R \mid x : S, n_2 : \bullet}}{[\![\mathcal{F}'_{\mathcal{G}}(P)]\!]_h = (\nu xy)(y(n_1) \mid \overline{x}(z).[z \leftrightarrow n_2]) \vdash_{\mathtt{H}} n_2 : \bullet}$$

As $\neg\text{NBR}(\mathcal{F}'_{\triangledown}(P)) \implies \mathcal{F}_{\triangledown}(P) = \{\mathcal{F}'_{\triangledown}(P)\}$.

Observe that:

$$[\![P]\!]_h \xrightarrow{\tau} (\nu xy)(\nu n_1 z)[z \leftrightarrow n_2] \xrightarrow{[z \leftrightarrow n_2]} (\nu xy)(\nu n_1 z)\mathbf{0} \equiv \mathbf{0}$$

Let $[\![\mathcal{F}_{\mathcal{G}}(P)]\!]_h \triangleq P^\bullet = (\nu xy)(y(n_1) \mid \overline{x}(z).[z \leftrightarrow n_2])$.

Then:

$$P^\bullet \xrightarrow{\tau} (\nu xy)(\nu n_1 z)(\mathbf{0} \mid [z \leftrightarrow n_2]) \equiv (\nu xy)(\nu n_1 z)[z \leftrightarrow n_2] \xrightarrow{[z \leftrightarrow n_2]} (\nu xy)(\nu n_1 z)\mathbf{0} \equiv \mathbf{0}$$

These are the only possible transitions that can occur for $[\![P]\!]_h, P^\bullet$. Thus, we can see the translation maintained the possible computations (i.e: transitions) and did not create any new ones.

Example 5.2.2. (Unrelated Names) This example shows a more complex translation, where there are actions that are unrelated to the non-blocking actions being moved. Consider:

$$P = (\boldsymbol{\nu} xy)(w(v).\overline{x}\langle n_1\rangle.y(n_2).w(n_3).v(n_4))$$

Let $R = \bullet \parr \bullet, S = \bullet \otimes \bullet$. Observe that $S = \overline{R}$.

Now consider the following proof $\triangledown$ of $[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{0} \vdash_{\mathtt{H}} y:\bullet, n_2:\bullet \mid w:\bullet, n_3:\bullet, v:\bullet, n_4:\bullet, x:\bullet \triangleq \mathcal{G}_5}{v(n_4) \vdash_{\mathtt{H}} y:\bullet, n_2:\bullet \mid w:\bullet, n_3:\bullet, v:R, x:\bullet \triangleq \mathcal{G}_4}}{w(n_3).v(n_4) \vdash_{\mathtt{H}} y:\bullet, n_2:\bullet \mid w:R, v:R, x:\bullet \triangleq \mathcal{G}_3}}{y(n_2).w(n_3).v(n_4) \vdash_{\mathtt{H}} y:R \mid w:R, v:R, x:\bullet \triangleq \mathcal{G}_2} \quad [z \leftrightarrow n_1] \vdash_{\mathtt{H}} z:\bullet, n_1:\bullet}{[z \leftrightarrow n_1] \mid y(n_2).w(n_3).v(n_4) \vdash_{\mathtt{H}} z:\bullet, n_1:\bullet \mid y:R \mid w:R, v:R, x:\bullet}}{\overline{x}(z).([z \leftrightarrow n_1] \mid y(n_2).w(n_3).v(n_4)) \vdash_{\mathtt{H}} w:R, v:R, x:S, n_1:\bullet \mid y:R \triangleq \mathcal{G}_1}}{w(v).\overline{x}(z).([z \leftrightarrow n_1] \mid y(n_2).w(n_3).v(n_4)) \vdash_{\mathtt{H}} w:R \parr R, x:S, n_1:\bullet \mid y:R}}{[\![P]\!]_h = (\boldsymbol{\nu} xy)(w(v).\overline{x}(z).([z \leftrightarrow n_1] \mid y(n_2).w(n_3).v(n_4))) \vdash_{\mathtt{H}} w:R \parr R, n_1:\bullet \triangleq \mathcal{G}}$$

Then:

$$\begin{aligned}
\mathcal{F}'_{\triangledown}(P) &= (\boldsymbol{\nu} xy)(w(v).(\mathcal{R}^{\{w,v,x,n_1\}}_{\triangledown_1}(\overline{x}\langle n_1\rangle.y(n_2).w(n_3).v(n_4)) \mid \mathcal{K}^{\{w,v,x,n_1\}}_{\triangledown_1}(\overline{x}\langle n_1\rangle.y(n_2).w(n_3).v(n_4)))) \\
&= (\boldsymbol{\nu} xy)(w(v).(\mathcal{R}^{\{w,v,x\}}_{\triangledown_2}(y(n_2).w(n_3).v(n_4)) \mid \overline{x}\langle n_1\rangle.\mathcal{K}^{\{w,v,x\}}_{\triangledown_2}(y(n_2).w(n_3).v(n_4)))) \\
&= (\boldsymbol{\nu} xy)(w(v).(y(n_2).\mathcal{R}^{\{w,v,x\}}_{\triangledown_3}(w(n_3).v(n_4)) \mid \overline{x}\langle n_1\rangle.\mathcal{K}^{\{w,v,x\}}_{\triangledown_3}(w(n_3).v(n_4)))) \\
&= (\boldsymbol{\nu} xy)(w(v).(y(n_2).\mathcal{R}^{\{w,v,x,n_3\}}_{\triangledown_4}(v(n_4)) \mid \overline{x}\langle n_1\rangle.w(n_3).\mathcal{K}^{\{w,v,x,n_3\}}_{\triangledown_4}(v(n_4)))) \\
&= (\boldsymbol{\nu} xy)(w(v).(y(n_2) \mid \overline{x}\langle n_1\rangle.w(n_3).v(n_4)))
\end{aligned}$$

As $\neg\text{NBR}(\mathcal{F}'_{\triangledown}(P)) \implies \mathcal{F}_{\triangledown}(P) = \{\mathcal{F}'_{\triangledown}(P)\}$.

Let us now show the transitions for $[\![P]\!]_h$

$$\begin{aligned}
[\![P]\!]_h &\xrightarrow{w(v)} (\boldsymbol{\nu} xy)(\overline{x}(z).([z \leftrightarrow n_1] \mid y(n_2).w(n_3).v(n_4))) \\
&\xrightarrow{\tau} (\boldsymbol{\nu} xy)(\boldsymbol{\nu} zn_2)([z \leftrightarrow n_1] \mid w(n_3).v(n_4)) \\
[\![P]\!]_h &\xrightarrow{\tau} (\boldsymbol{\nu} xy)(\boldsymbol{\nu} zn_2)(w(v).([z \leftrightarrow n_1] \mid w(n_3).v(n_4)))
\end{aligned}$$

Let $[\![\mathcal{F}'_{\mathcal{G}}(P)]\!]_h \triangleq P^{\bullet} = (\boldsymbol{\nu} xy)(w(v).(y(n_2) \mid \overline{x}(n_2).([z \leftrightarrow n_1] \mid w(n_3).v(n_4))))$. Then, looking at its transitions:

$$\begin{aligned}
P^{\bullet} &\xrightarrow{w(v)} (\boldsymbol{\nu} xy)(y(n_2) \mid \overline{x}(n_2).([z \leftrightarrow n_1] \mid w(n_3).v(n_4))) \\
&\xrightarrow{\tau} (\boldsymbol{\nu} xy)(\boldsymbol{\nu} zn_2)(\mathbf{0} \mid [z \leftrightarrow n_1] \mid w(n_3).v(n_4)) \equiv (\boldsymbol{\nu} xy)(\boldsymbol{\nu} zn_2)([z \leftrightarrow n_1] \mid w(n_3).v(n_4)) \\
P^{\bullet} &\xrightarrow{\tau} (\boldsymbol{\nu} xy)(\boldsymbol{\nu} zn_2)(w(v).(\mathbf{0} \mid [z \leftrightarrow n_1] \mid w(n_3).v(n_4))) \equiv (\boldsymbol{\nu} xy)(\boldsymbol{\nu} zn_2)(w(v).([z \leftrightarrow n_1] \mid w(n_3).v(n_4)))
\end{aligned}$$

We do not continue the subsequent transitions, since it is obvious that the generated sequences will be present when transition from $P^{\bullet}$ or $[\![P]\!]_h$.

Thus, we can see the translation maintained the possible computations (i.e: transitions) and did not create any new ones.

Example 5.2.3. (Same Process, Different Translation) This example shows a translation of the process in Example 5.2.2, but using a different proof of $P \vdash_{\mathtt{H}} \mathcal{G}$.

$$P = (\nu xy)(w(v).\overline{x}\langle n_1\rangle.y(n_2).w(n_3).v(n_4))$$

Let $R = \bullet \mathbin{⅋} \bullet, S = \bullet \otimes \bullet$. Observe that $S = \overline{R}$.

Now consider the following proof $\triangledown$ of $[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{0} \vdash_{\mathtt{H}} w : \bullet, n_3 : \bullet, v : \bullet, n_4 : \bullet, y : \bullet, n_2 : \bullet \mid x : \bullet \triangleq \mathcal{G}_5}{v(n_4) \vdash_{\mathtt{H}} w : \bullet, n_3 : \bullet, v : R, y : \bullet, n_2 : \bullet \mid x : \bullet \triangleq \mathcal{G}_4}}{w(n_3).v(n_4) \vdash_{\mathtt{H}} w : R, v : R, y : \bullet, n_2 : \bullet \mid x : \bullet \triangleq \mathcal{G}_3}}{y(n_2).w(n_3).v(n_4) \vdash_{\mathtt{H}} w : R, v : R, y : R \mid x : \bullet \triangleq \mathcal{G}_2} \quad [z \leftrightarrow n_1] \vdash_{\mathtt{H}} z : \bullet, n_1 : \bullet}{[z \leftrightarrow n_1] \mid y(n_2).w(n_3).v(n_4) \vdash_{\mathtt{H}} z : \bullet, n_1 : \bullet \mid w : R, v : R, , y : R \mid x : \bullet}}{\overline{x}(z).([z \leftrightarrow n_1] \mid y(n_2).w(n_3).v(n_4)) \vdash_{\mathtt{H}} x : S, n_1 : \bullet \mid w : R, v : R, y : R \triangleq \mathcal{G}_1}}{w(v).\overline{x}(z).([z \leftrightarrow n_1] \mid y(n_2).w(n_3).v(n_4)) \vdash_{\mathtt{H}} x : S, n_1 : \bullet \mid w : R \mathbin{⅋} R, y : R}}{[\![P]\!]_h = (\nu xy)(w(v).\overline{x}(z).([z \leftrightarrow n_1] \mid y(n_2).w(n_3).v(n_4))) \vdash_{\mathtt{H}} w : R \mathbin{⅋} R, n_1 : \bullet \triangleq \mathcal{G}}$$

Then:

$$\begin{aligned}
\mathcal{F}'_{\triangledown}(P) &= (\nu xy)(w(v).(\mathcal{R}^{\{x,n_1\}}_{\triangledown_1}(\overline{x}\langle n_1\rangle.y(n_2).w(n_3).v(n_4)) \mid \mathcal{K}^{\{x,n_1\}}_{\triangledown_1}(\overline{x}\langle n_1\rangle.y(n_2).w(n_3).v(n_4)))) \\
&= (\nu xy)(w(v).(\mathcal{R}^{\{x\}}_{\triangledown_2}(y(n_2).w(n_3).v(n_4)) \mid \overline{x}\langle n_1\rangle.\mathcal{K}^{\{x\}}_{\triangledown_2}(y(n_2).w(n_3).v(n_4)))) \\
&= (\nu xy)(w(v).(y(n_2).\mathcal{R}^{\{x\}}_{\triangledown_3}(w(n_3).v(n_4)) \mid \overline{x}\langle n_1\rangle.\mathcal{K}^{\{x\}}_{\triangledown_3}(w(n_3).v(n_4)))) \\
&= (\nu xy)(w(v).(y(n_2).w(n_3).\mathcal{R}^{\{x\}}_{\triangledown_4}(v(n_4)) \mid \overline{x}\langle n_1\rangle.\mathcal{K}^{\{x\}}_{\triangledown_4}(v(n_4)))) \\
&= (\nu xy)(w(v).(y(n_2).w(n_3).v(n_4) \mid \overline{x}\langle n_1\rangle))
\end{aligned}$$

As $\neg\mathrm{NBR}(\mathcal{F}'_{\triangledown}(P)) \implies \mathcal{F}_{\triangledown}(P) = \{\mathcal{F}'_{\triangledown}(P)\}$.

We do not present the transitions that $[\![P]\!]_h$ can follow, as they are the same from the previous example.

Let $[\![\mathcal{F}'_{\mathcal{G}}(P)]\!]_h \triangleq P^{\bullet} = (\nu xy)(w(v).(y(n_2).w(n_3).v(n_4) \mid \overline{x}(z).[z \leftrightarrow n_1]))$. Then looking at its transitions.

Then

$$\begin{aligned}
P^{\bullet} &\xrightarrow{w(v)} (\nu xy)(y(n_2).w(n_3).v(n_4) \mid \overline{x}(z).[z \leftrightarrow n_1]) \\
&\xrightarrow{\tau} (\nu xy)(\nu z n_2)(w(n_3).v(n_4) \mid [z \leftrightarrow n_1]) \\
P^{\bullet} &\xrightarrow{\tau} (\nu xy)(\nu z n_2)(w(v).(w(n_3).v(n_4) \mid [z \leftrightarrow n_1])) \equiv (\nu xy)(\nu z n_1)(w(v).([z \leftrightarrow n_1] \mid w(n_3).v(n_4)))
\end{aligned}$$

We do not continue the subsequent transitions, since it is obvious that the generated sequences will be present when transition from $P^{\bullet}$ or $[\![P]\!]_h$.

Thus, we can see the translation maintained the possible computations (i.e: transitions) and did not create any new ones.

So it is reassuring to observe that using a different proof for $\mathcal{F}'_{\cdot}(\cdot)$ does not effect the transition sequences that can be generated.

Example 5.2.4. (Movement of the restriction operator) This example shows how restriction might be indirectly moved. Consider:

$$P = (\boldsymbol{\nu} xy)(\boldsymbol{\nu} wv)\overline{x}\langle n_1\rangle.y(n_2).(\overline{v}\langle n_3\rangle \mid w(n_4))$$

Let $R = \bullet \parr \bullet, S = \bullet \otimes \bullet$. Observe that $S = \overline{R}$.

We can have:

$$\dfrac{\dfrac{\dfrac{[z_1 \leftrightarrow n_1] \vdash_{\mathtt{H}} z_1 : \bullet, n_1 : \bullet}{[z_1 \leftrightarrow n_1] \vdash_{\mathtt{H}}\mid z_1 : \bullet, n_1 : \bullet} \quad \dfrac{\dfrac{\dfrac{\dfrac{[z_2 \leftrightarrow n_3] \vdash_{\mathtt{H}} z_2 : \bullet, n_3 : \bullet}{[z_2 \leftrightarrow n_3] \vdash_{\mathtt{H}} z_2 : \bullet, n_3 : \bullet \mid v : \bullet}}{\overline{v}(z_2).[z_2 \leftrightarrow n_3] \vdash_{\mathtt{H}} v : S, n_3 : \bullet \triangleq \mathcal{G}_5^l} \quad \dfrac{\mathbf{0} \vdash_{\mathtt{H}} x : \bullet, w : \bullet, n_4 : \bullet \mid y : \bullet, n_2 : \bullet}{w(n_4) \vdash_{\mathtt{H}} x : \bullet, w : R \mid y : \bullet, n_2 : \bullet \triangleq \mathcal{G}_5^r}}{\overline{v}(z_2).[z_2 \leftrightarrow n_3] \mid w(n_4) \vdash_{\mathtt{H}} v : S, n_3 : \bullet \mid x : \bullet, w : R \mid y : \bullet, n_2 : \bullet \triangleq \mathcal{G}_4}}{y(n_2).(\overline{v}(z_2).[z_2 \leftrightarrow n_3] \mid w(n_4)) \vdash_{\mathtt{H}} v : S, n_3 : \bullet \mid x : \bullet, w : R \mid y : R \triangleq \mathcal{G}_3}}{\dfrac{[z_1 \leftrightarrow n_1] \mid y(n_2).(\overline{v}(z_2).[z_2 \leftrightarrow n_3] \mid w(n_4)) \vdash_{\mathtt{H}} z_1 : \bullet, n_1 : \bullet \mid v : S, n_3 : \bullet \mid x : \bullet, w : R \mid y : R}{\dfrac{\overline{x}(z_1).([z_1 \leftrightarrow n_1] \mid y(n_2).(\overline{v}(z_2).[z_2 \leftrightarrow n_3] \mid w(n_4))) \vdash_{\mathtt{H}} x : S, w : R, n_1 : \bullet \mid v : S, n_3 : \bullet \mid y : R \triangleq \mathcal{G}_2}{\dfrac{(\boldsymbol{\nu} wv)\overline{x}(z_1).([z_1 \leftrightarrow n_1] \mid y(n_2).(\overline{v}(z_2).[z_2 \leftrightarrow n_3] \mid w(n_4))) \vdash_{\mathtt{H}} x : S, n_1 : R, n_3 : \bullet \mid y : R \triangleq \mathcal{G}_1}{[\![P]\!]_h = (\boldsymbol{\nu} xy)(\boldsymbol{\nu} wv)\overline{x}(z_1).([z_1 \leftrightarrow n_1] \mid y(n_2).(\overline{v}(z_2).[z_2 \leftrightarrow n_3] \mid w(n_4))) \vdash_{\mathtt{H}} n_1 : \bullet, n_3 : \bullet \triangleq \mathcal{G}}}}}$$

Then:

$$\begin{aligned}
\mathcal{F}'_{\triangledown}(P) &= (\boldsymbol{\nu} xy)(\mathcal{R}^{\{x,n_1\}}_{\triangledown_1}((\boldsymbol{\nu} wv)\overline{x}\langle n_1\rangle.y(n_2).(\overline{v}\langle n_3\rangle \mid w(n_4))) \mid \mathcal{K}^{\{x,n_1\}}_{\triangledown_1}((\boldsymbol{\nu} wv)\overline{x}\langle n_1\rangle.y(n_2).(\overline{v}\langle n_3\rangle \mid w(n_4)))) \\
&= (\boldsymbol{\nu} xy)(\mathcal{R}^{\{x,n_1,w,v\}}_{\triangledown_2}(\overline{x}\langle n_1\rangle.y(n_2).(\overline{v}\langle n_3\rangle \mid w(n_4))) \mid (\boldsymbol{\nu} wv)\mathcal{K}^{\{x,n_1,w,v\}}_{\triangledown_2}(\overline{x}\langle n_1\rangle.y(n_2).(\overline{v}\langle n_3\rangle \mid w(n_4)))) \\
&= (\boldsymbol{\nu} xy)(\mathcal{R}^{\{x,w,v\}}_{\triangledown_3}(y(n_2).(\overline{v}\langle n_3\rangle \mid w(n_4))) \mid (\boldsymbol{\nu} wv)\overline{x}\langle n_1\rangle.\mathcal{K}^{\{x,w,v\}}_{\triangledown_3}(y(n_2).(\overline{v}\langle n_3\rangle \mid w(n_4)))) \\
&= (\boldsymbol{\nu} xy)(y(n_2).\mathcal{R}^{\{x,w,v\}}_{\triangledown_4}((\overline{v}\langle n_3\rangle \mid w(n_4))) \mid (\boldsymbol{\nu} wv)\overline{x}\langle n_1\rangle.\mathcal{K}^{\{x,w,v\}}_{\triangledown_4}((\overline{v}\langle n_3\rangle \mid w(n_4)))) \\
&= (\boldsymbol{\nu} xy)(y(n_2).(\mathcal{R}^{\{x,v\}}_{\triangledown_5^l}(\overline{v}\langle n_3\rangle) \mid \mathcal{R}^{\{x,w\}}_{\triangledown_5^r}(w(n_4))) \mid (\boldsymbol{\nu} wv)\overline{x}\langle n_1\rangle.(\mathcal{K}^{\{x,v\}}_{\triangledown_5^l}(\overline{v}\langle n_3\rangle) \mid \mathcal{K}^{\{x,w\}}_{\triangledown_5^r}(w(n_4)))) \\
&= (\boldsymbol{\nu} xy)(y(n_2) \mid (\boldsymbol{\nu} wv)\overline{x}\langle n_1\rangle.(\overline{v}\langle n_3\rangle \mid w(n_4)))
\end{aligned}$$

As $\neg\mathrm{NBR}(\mathcal{F}'_{\triangledown}(P)) \implies \mathcal{F}_{\triangledown}(P) = \mathcal{F}'_{\triangledown}(P)$.

Now observe that:

$$[\![P]\!]_h \xrightarrow{\tau} (\boldsymbol{\nu} xy)(\boldsymbol{\nu} wv)(\boldsymbol{\nu} z_1 n_2)([z_1 \leftrightarrow n_1] \mid \overline{v}(z_2).[z_2 \leftrightarrow n_3] \mid w(n_4))$$

Let $[\![\mathcal{F}'_{\mathcal{G}}(P)]\!]_h \triangleq P^{\bullet} = (\boldsymbol{\nu} xy)(y(n_2) \mid (\boldsymbol{\nu} wv)\overline{x}(z_1).([z_1 \leftrightarrow n_1] \mid \overline{v}(z_2).[z_2 \leftrightarrow n_3] \mid w(n_4)))$. Then:

$$\begin{aligned}
P^{\bullet} &\xrightarrow{\tau} (\boldsymbol{\nu} xy)(\boldsymbol{\nu} z_1 n_2)(\mathbf{0} \mid (\boldsymbol{\nu} wv)([z_1 \leftrightarrow n_1] \mid \overline{v}(z_2).[z_2 \leftrightarrow n_3] \mid w(n_4))) \\
&\equiv (\boldsymbol{\nu} xy)(\boldsymbol{\nu} z_1 n_2)(\boldsymbol{\nu} wv)([z_1 \leftrightarrow n_1] \mid \overline{v}(z_2).[z_2 \leftrightarrow n_3] \mid w(n_4)) \\
&\equiv (\boldsymbol{\nu} xy)(\boldsymbol{\nu} wv)(\boldsymbol{\nu} z_1 n_2)([z_1 \leftrightarrow n_1] \mid \overline{v}(z_2).[z_2 \leftrightarrow n_3] \mid w(n_4))
\end{aligned}$$

We do not continue the subsequent transitions, since it is obvious that the generated sequences will be present when transition from $P^{\bullet}$ or $[\![P]\!]_h$.

Thus, we can see the translation maintained the possible computations (i.e: transitions) and did not create any new ones.

5.3 Properties of the translation

We first discuss properties associated with the auxiliary translation, $\mathcal{F}'_{\cdot}(\cdot)$, and then those of the main translation $\mathcal{F}_{\cdot}(\cdot)$. The following Lemmas will be useful for proving properties of the main translation.

Lemma 5.3.1. *($\mathcal{F}'_{\cdot}(\cdot)$ is type preserving)*

$$[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G} \wedge P \notin \mathcal{K} \implies [\![\mathcal{F}'_{\nabla}(P)]\!]_h \vdash_{\mathtt{H}} \mathcal{G}$$

for all proofs, ∇, of $[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}$.

Proof. By case analysis on the definition of $\mathcal{F}'_{\cdot}(\cdot)\cdot$ and by observing that if $[\![C[\pi.P']]\!]_h \vdash_{\mathtt{H}} \mathcal{G}^*$ then

$$[\![\mathcal{R}^{N_i}_{\nabla^*}(C[\pi.P']) \mid \mathcal{K}^{N_i}_{\nabla^*}(C[\pi.P'])]\!]_h \vdash_{\mathtt{H}} \mathcal{G}^*$$

for $i \in \{1, 2\}$, where $C, \pi, P', \nabla^*, N_i$ are as defined in the definition of $\mathcal{F}'_{\cdot}(\cdot)\cdot$ (Definition 5.1.3).

See §B.1 for the proof. □

The following two lemmas will be useful for the proof of Lemma 5.3.3. This shows that the order of actions is maintained by $\mathcal{F}'_{\cdot}(\cdot)$.

We split this property into two properties / lemmas, to precisely refer to this notion.

Lemma 5.3.2. *($\mathcal{F}'_{\cdot}(\cdot)$ maintains order of actions)*

$$\mathcal{F}'_{\nabla}(P) = D_1[\alpha.D_2[\beta.Q]] \implies P = D'_1[\alpha.D'_2[\beta.Q']]$$

Proof. It suffices to show that this property holds for $\mathcal{K}_{\cdot}(\cdot)$ and $\mathcal{R}_{\cdot}(\cdot)$.

That is, we seek to prove:

$$\begin{aligned} \mathcal{K}^{N}_{\nabla}(P) = D_1[\alpha.D_2[\beta.Q]] &\implies P = D'_1[\alpha.D'_2[\beta.Q']] \\ \mathcal{R}^{N}_{\nabla}(P) = D_1[\alpha.D_2[\beta.Q]] &\implies P = D'_1[\alpha.D'_2[\beta.Q']] \end{aligned}$$

This is proven via case analysis over the definition of $\mathcal{K}_{\cdot}(\cdot)$ and $\mathcal{R}_{\cdot}(\cdot)$ respectively. □

Corollary 5.3.1.

$$\mathcal{F}'_{\nabla}(P) = D_1[\alpha.D_2[b \triangleright \{l_i : P_i\}_{i\in I}]] \implies P = D'_1[\alpha.D'_2[b \triangleright \{l_i : P_i\}_{i\in I}]]$$

Proof. The proof is the same as for Lemma 5.3.2, except we change the properties we seek to prove for $\mathcal{K}_{\cdot}(\cdot), \mathcal{R}_{\cdot}(\cdot)$. However, the method to prove these properties are the same. □

Lemma 5.3.3. *($\mathcal{F}'_{\cdot}(\cdot)$ reduces number of non-blocking reliant channels)*

$$[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G} \wedge P \notin \mathcal{K} \implies |\mathrm{NBC}([\![\mathcal{F}'_{\nabla}(P)]\!]_h)| \leq |\mathrm{NBC}(P)| - 1$$

for all proofs, ∇, of $[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}$.

Proof. Let $\triangledown$ be any proof of $[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}$.

As $[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G} \wedge P \notin \mathcal{K}$, we have that $P \in \mathcal{H}^\circ$, meaning that $\mathrm{NBR}(P)$, by Theorem 4.2.1. Thus:

$$\exists D, C, P' : P = D[(\nu xy)C[\pi.P']]$$

where $\mathrm{sub}(\pi) = x \wedge y \in \mathrm{fn}(P')$ (or $\mathrm{sub}(\pi) = y \wedge x \in \mathrm{fn}(P')$, but the argument follows analogously), with $(x, y) \in \mathrm{NBC}(P)$.

However, observe that $(x, y) \notin \mathrm{NBC}(\mathcal{F}'_{\mathcal{G}}(P))$.

We have $x \in \mathrm{fn}(\mathcal{K}^{N_1}_{\triangledown^*}(C[\pi.P'])), x \notin \mathrm{fn}(\mathcal{R}^{N_1}_{\triangledown^*}(C[\pi.P'])), y \in \mathrm{fn}(\mathcal{R}^{N_1}_{\triangledown^*}(C[\pi.P'])), y \notin \mathrm{fn}(\mathcal{K}^{N_1}_{\triangledown^*}(C[\pi.P']))$ via structural induction over $C[\pi.P']$.

Thus, observe that $(x, y) \notin \mathrm{NBC}(\mathcal{R}^{N_1}_{\triangledown^*}(C[\pi.P']) \mid \mathcal{K}^{N_1}_{\triangledown^*}(C[\pi.P']))$.

Then x, y are bound and thus, can never prefix each other again, due to Barendregt's convention. And so, $(x, y) \notin \mathrm{NBC}(D[(\nu xy)(\mathcal{R}^{N_1}_{\triangledown^*}(C[\pi.P']) \mid \mathcal{K}^{N_1}_{\triangledown^*}(C[\pi.P']))])$

Furthermore, $\mathrm{NBC}(\mathcal{F}'_{\triangledown}(P)) \subseteq \mathrm{NBC}(P)$. We will prove this via contradiction. Suppose

$$\exists (a, b) \in \mathrm{NBC}(\mathcal{F}'_{\triangledown}(P)) : (a, b) \notin \mathrm{NBC}(P)$$

This means that

$$\exists D_t, C_t, P_t : \mathcal{F}'_{\triangledown}(P) = D_t[(\nu ab)C_t[\alpha.P_t]]$$

where, without loss of generality, we have $\mathrm{sub}(\alpha) = a$ and $b \in \mathrm{fn}(P_t)$. For our assumption to be true, we cannot have

$$\exists D_u, C_u, P_u : P = D_u[(\nu ab)C_u[\alpha.P_u]]$$

where $b \in \mathrm{fn}(P_u)$.

Now $b \in P_t \implies P_t = D_1[\beta.R]$ (where $\mathrm{sub}(\beta) = b$) or $P_t = D_1[b \triangleright \{l_i : P_i\}_{i \in I}]$.

However, if $\alpha.D_1[\beta.R]$ (or $\alpha.D_1[b \triangleright \{l_i : P_i\}_{i \in I}]$) is a subprocess in $\mathcal{F}'_{\triangledown}(P)$, we must have $\alpha.D_2[\beta.Q]$ (or $\alpha.D_2[b \triangleright \{l_i : P_i\}_{i \in I}]$) be a subprocess in P, via Lemma 5.3.2 (or Corollary 5.3.1).

Furthermore, if $\mathcal{F}'_{\triangledown}(P) = D[(\nu ab)Q]$ then $P = D'[(\nu ab)Q']$, as $(\nu ab)Q^*$ must be a subprocess in either $\mathcal{K}^{\triangledown^*}_{N_i}(C[\pi.P'])$ or $\mathcal{R}^{\triangledown^*}_{N_i}(C[\pi.P'])$, by definition, for the appropriate Q^*.

Then, by Barendregt's variable convention, we must have $P = D_u[(\nu ab)C_u[\alpha.P_u]]$, with $b \in \mathrm{fn}(P_u)$, as α and β (or $b \triangleright \{l_i : P_i\}_{i \in I}$) cannot appear outside of Q'.

Thus, $\mathrm{NBC}(\mathcal{F}'_{\triangledown}(P)) \subsetneq \mathrm{NBC}(P)$.

Therefore, $|\mathrm{NBC}([\![\mathcal{F}'_{\triangledown}(P)]\!]_h)| \leq |\mathrm{NBC}(P)| - 1$. □

We now discuss properties of our main translation.

As our translation is algorithmic in nature, we must show that it is terminating (in the sense that it does not infinitely apply $\mathcal{F}'$).

Lemma 5.3.4. *(Translation is terminating)*

$$[\![P]\!]_h \vdash_{H} \mathcal{G} \wedge P \notin \mathcal{K} \implies \exists m : \forall P^\bullet \in \mathcal{F}_{\triangledown_1}(P) : P^\bullet = \mathcal{F}'_{\triangledown_m}(\mathcal{F}'_{\triangledown_{m-1}}(\cdots(\mathcal{F}'_{\triangledown_1}(P))))$$

for some proofs $\triangledown_i$ *of* $\mathcal{F}'_{\triangledown_{i-1}}(\mathcal{F}'_{\triangledown_{m-1}}(\cdots(\mathcal{F}'_{\triangledown_1}(P)))) \vdash_{H} \mathcal{G}$

Proof. Let $\triangledown_1$ be any proof of $[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}$. Let $P^\bullet \in \mathcal{F}_{\triangledown_1}(P)$.

We will have $m \leq |\mathrm{NBC}(P)|$.

When $|\mathrm{NBC}(P)| = 0$, we have that $\neg\mathrm{NBR}(P)$ and so $P^\bullet = P$. Thus $m = |\mathrm{NBC}(P)| = 0$.

Otherwise, observe that $\neg\mathrm{NBR}(P)$ and so $\exists D, C, P' : P = D[(\nu xy)C[\pi.P']]$ where $\mathrm{sub}(\pi) = x \wedge y \in \mathrm{fn}(P')$ (or $\mathrm{sub}(\pi) = y \wedge x \in \mathrm{fn}(P')$, but the argument follows analogously).

Thus, $P^\bullet = \mathcal{F}_{\triangledown_2}(\mathcal{F}'_{\triangledown_1}(P))$, for some proof $\triangledown_2$ of $\mathcal{F}'_{\triangledown_1}(P) \vdash_{\mathtt{H}} \mathcal{G}$ (which must exist due to Lemma 5.3.1). But observe that $|\mathrm{NBC}(\mathcal{F}'_{\triangledown_1}(P))| \leq m-1$, via Lemma 5.3.3

Thus, by the same argument, we must have $P^\bullet = \mathcal{F}_{\triangledown_3}(\mathcal{F}'_{\triangledown_2}(\mathcal{F}'_{\triangledown_1}(P)))$, with $\mathrm{NBC}(\mathcal{F}'_{\triangledown_2}(\mathcal{F}'_{\triangledown_1}(P))) \leq m-2$

Repeating this argument $m-2$ more times, we have

$$P^\bullet = \mathcal{F}_{\triangledown_{m+1}}(\mathcal{F}'_{\triangledown_m}(\cdots(\mathcal{F}'_{\triangledown_1}(P))))$$

where

$$|\mathrm{NBC}(\mathcal{F}'_{\triangledown_m}(\cdots(\mathcal{F}'_{\triangledown_1}(P))))| \leq 0$$

which means that $|\mathrm{NBC}(\mathcal{F}'_{\triangledown_m}(\cdots(\mathcal{F}'_{\triangledown_1}(P))))| = 0$.

But then $P^\bullet = \mathcal{F}_{\triangledown}(\mathcal{F}'_{\triangledown_m}(\cdots(\mathcal{F}'_{\triangledown_1}(P)))) = \mathcal{F}'_{\triangledown_m}(\cdots(\mathcal{F}'_{\triangledown_1}(P)))$, and we have concluded our proof. ◻

We then prove that our translation results in a process that is not reliant on non-blocking semantics, a key goal of our translation.

Corollary 5.3.2. *(Translation fixes all channels)*

$$[\![P]\!]_h \vdash_{H} \mathcal{G} \wedge P \notin \mathcal{K} \implies \forall P^\bullet \in \mathcal{F}_{\triangledown}(P) : \neg\mathrm{NBR}(P^\bullet))$$

for all proofs $\triangledown$ *of* $[\![P]\!]_h \vdash_{H} \mathcal{G}$

Proof. Let $\triangledown$ be some proof of $[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}$. Let $P^\bullet \in \mathcal{F}_{\triangledown}(P)$.

By the proof of Lemma 5.3.4, we have that $P^\bullet = \mathcal{F}'_{\triangledown_{m-1}}(\cdots(\mathcal{F}'_{\triangledown_1}(P)))$, with $\mathrm{NBC}(P^\bullet) = 0$.

Thus, $\neg\mathrm{NBR}(P^\bullet)$, which concludes our proof. ◻

We then prove our translation is type preserving, in that the resulting translation is typable under the same hypercontext as the untranslated process.

Lemma 5.3.5. *(Translation is type preserving)*

$$[\![P]\!]_h \vdash_{H} \mathcal{G} \wedge P \notin \mathcal{K} \implies \forall P^\bullet \in \mathcal{F}_{\triangledown}(P) : [\![P^\bullet]\!]_h \vdash_{H} \mathcal{G}$$

for all proofs $\triangledown$ of $[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}$.

Proof. Let $\triangledown$ be some proof of $[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}$. Let $P^\bullet \in \mathcal{F}_\triangledown(P)$.

Observe that $P^\bullet = \mathcal{F}'_{\triangledown_m}(\mathcal{F}'_{\triangledown_{m-1}}(\cdots(\mathcal{F}'_{\triangledown_1}(P))))$, via Lemma 5.3.4. By Lemma 5.3.1 and induction on the length of the composition, we have our thesis. □

Finally, using the aforementioned results, we can show our translation achives its goal: translating processes in $\mathcal{H}^\circ$ to $\mathcal{H}^\bullet$.

Theorem 5.3.1. *(Translation guarantees deadlock-freedom under blocking semantics)*

$$[\![P]\!]_h \vdash_{H} \mathcal{G} \wedge P \notin \mathcal{K} \implies \forall P^\bullet \in \mathcal{F}_\triangledown(P) : [\![P^\bullet]\!]_h \vdash_{H} \mathcal{G} \wedge P^\bullet \in \mathcal{H}^\bullet$$

for all proofs $\triangledown$ *of* $[\![P]\!]_h \vdash_{H} \mathcal{G}$.

Proof. Let $\triangledown$ be some proof of $[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}$. Let $P^\bullet \in \mathcal{F}_\triangledown(P)$.

By Lemma 5.3.5, we have that $P^\bullet \in \mathcal{H}$. By Corollary 5.3.2 and Theorem 4.2.1 we have that $P^\bullet \notin \mathcal{H}^\circ$ and so $P^\bullet \notin \mathcal{H} \vee P^\bullet \notin \mathcal{K}$. Thus we have that $P^\bullet \in \mathcal{K}$ and so we have $P^\bullet \in \mathcal{H}^\bullet$. □

However, these properties merely show that our translation satisfies the basic quality of session fidelity. As discussed in §2, we must show translation does not lose any behaviour captured by a process in $\mathcal{H}^\circ$. Thus we must establish an operational correspondence property, for our translation.

We have not been able to prove any results on operational correspondence, due to the time afforded to this project, but we will present operational conjectures motivated by our examples from §5.2.

Furthermore, we will present two conjectures. This is because we are dealing with two different semantics, which will have a consequence on the behaviour that our translation introduces or preserves. Thus, we present a conjecture with regards to blocking semantics and another with regards to non-blocking / self-synchronisation semantics.

We first present a conjecture regarding operational correspondence under the blocking semantics:

Conjecture 5.3.1. *(Completeness under blocking semantics)*

Let $[\![P]\!]_h \vdash_{H} \mathcal{G}$. *Then:*

- $P \to^* Q \implies \exists Q^\bullet \in \mathcal{F}_{\triangledown_Q}(Q) : P^\bullet \to^* Q^\bullet$

for all $P^\bullet \in \mathcal{F}_{\triangledown_P}(P)$ *where:*

- $\triangledown_P$ *is any proof of* $[\![P]\!]_h \vdash_{H} \mathcal{G}$
- $\triangledown_Q$ *is a proof of* $[\![P]\!]_h \vdash_{H} \mathcal{Q}$

We suspect there is a completeness property in our translation but not a soundness property. This is because our translation takes some deadlocked processes and makes them deadlock-free, meaning that there might be a processes P where $P^\bullet \in \mathcal{F}_\triangledown(P)$ reduces, but P itself cannot reduce (as it deadlocks). However, any process that would have an action under reduction semantics should occur under the translated process, since we merely add parallelism to the process.

We now present a conjecture regarding operational correspondence under the non-blocking semantics.

Conjecture 5.3.2. *(Operational Correspondence under non-blocking semantics)*

Let $[\![P]\!]_h \vdash_H \mathcal{G}$. *Then:*

- $[\![P]\!]_h \xrightarrow{l_1} \cdots \xrightarrow{l_n} [\![Q]\!]_h \implies \exists Q^\bullet \in \mathcal{F}_{\nabla_Q}(Q) : [\![P^\bullet]\!]_h \xrightarrow{l_1} \cdots \xrightarrow{l_n} [\![Q^\bullet]\!]_h$
- $[\![P^\bullet]\!]_h \xrightarrow{l_1} \cdots \xrightarrow{l_n} [\![Q]\!]_h \implies \exists P' : [\![P]\!]_h \xrightarrow{l_1} \cdots \xrightarrow{l_n} [\![P']\!]_h \wedge Q \in \mathcal{F}_{\nabla_{P'}}(P')$

For all $P^\bullet \in \mathcal{F}_\nabla(P)$ *where:*

- ∇_P *is any proof of* $[\![P]\!]_h \vdash_H \mathcal{G}$
- $\nabla_{P'}$ *is any proof of* $[\![P]\!]_h \vdash_H \mathcal{G}'$
- ∇_Q *is a proof of* $[\![P]\!]_h \vdash_H \mathcal{Q}$

We believe there is both a completeness and soundness property in our translation. The fact that the examples show new transition sequences lost / created gives confidence to this effect.

The key reasons we believe these properties hold is as follows:

- Our translation preserves the order in which actions are taken.
- Our translation does not "add" or "remove" process constructs.
- Parallelism does not affect the order in which actions must be taken. If we have a process $\alpha.\beta.P$ and the process $\alpha.P_1 \mid \beta.P_2$, both processes can have transition on either α or β, due to non-blocking semantics.

6 Discussion

In the following section, we discuss some insights gained in this work. Note that we often refer to the systems in [Wad12], [KMP19] as independent and different systems to our variant of these systems. For clarity and brevity, we refer to the linear fragment of [Wad12] as CP' and the linear fragment of [KMP19] as HCP'.

Also note that in CP', HCP', their syntax allows for communication without message-passing: $\overline{x}()$, $x()$. Lastly, note that CP' and HCP' use different notation for (non-message passing) bound output: $x[\;], x[y]$.

Non-blocking semantics do not yield new interactive behaviours A interesting consequence of our translation (if the operational correspondence holds true) is that non-blocking semantics do not yield behaviours that are not captured by blocking semantics. However, this should not be altogether that surprising, as using hypersequents does not yield an extension or different formulation of linear logic, just merely a different presentation. In this sense, the proofs in linear logic it captures (and thus the processes it induces) should not be significantly different.

Safe Cyclic Processes in HCP [DP22] conjectures that a of HCP and the dπ-calculus would enjoy the same results as a comparison between CP and the dπ-calculus, as HCP cannot capture the safe cyclic processes that the dπ-calculus captures. Thus, we would expect the process P in the proof of Lemama 4.2

would not be well-typed in HCP (after encoding). Thus, we should be able to use P to separate $\mathcal{H}^\bullet$ and $\mathcal{K}$. However, observe that it will-typed in HCP:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[z_1 \leftrightarrow x] \vdash_{\mathtt{H}} z_1 : \bullet, x : \bullet}{[z_1 \leftrightarrow x] \vdash_{\mathtt{H}} a_1 : \bullet, z_1 : \bullet, x : \bullet \mid a_2 : \bullet}}{\overline{a_2}(z_1).[z_1 \leftrightarrow x] \vdash_{\mathtt{H}} a_1 : \bullet, a_2 : S, x : \bullet}}{a_1(x).\overline{a_2}(z_1).[z_1 \leftrightarrow x] \vdash_{\mathtt{H}} a_1 : R, a_2 : S} \quad \dfrac{\dfrac{\dfrac{[n \leftrightarrow z_2] \vdash_{\mathtt{H}} n : \bullet, z_2 : \bullet}{[n \leftrightarrow z_2] \vdash_{\mathtt{H}} b_1 : \bullet, n : \bullet, z_2 : \bullet} \quad \dfrac{\mathbf{0} \vdash_{\mathtt{H}} b_2 : \bullet, z : \bullet}{b_2(z) \vdash_{\mathtt{H}} b_2 : R}}{[n \leftrightarrow z_2] \mid b_2(z) \vdash_{\mathtt{H}} b_1 : \bullet, n : \bullet, z_2 : \bullet \mid b_2 : R}}{\overline{b_1}(z_2).([n \leftrightarrow z_2] \mid b_2(z)) \vdash_{\mathtt{H}} b_1 : S, n : \bullet \mid b_2 : R}}{a_1(x).\overline{a_2}(z_1).[z_1 \leftrightarrow x] \mid \overline{b_1}(z_2).([n \leftrightarrow z_2] \mid b_2(z)) \vdash_{\mathtt{H}} a_1 : R, a_2 : S \mid b_1 : S, n : \bullet \mid b_2 : R}}{(\nu a_2 b_2)(a_1(x).\overline{a_2}(z_1).[z_1 \leftrightarrow x] \mid \overline{b_1}(z_2).([n \leftrightarrow z_2] \mid b_2(z))) \vdash_{\mathtt{H}} a_1 : R \mid b_1 : S, n : \bullet}}{[\![P]\!]_h = (\nu a_1 b_1)(\nu a_2 b_2)(a_1(x).\overline{a_2}(z_1).[z_1 \leftrightarrow x] \mid \overline{b_1}(z_2).([n \leftrightarrow z_2] \mid b_2(z))) \vdash_{\mathtt{H}} n : \bullet}$$

Furthermore, we do not believe this is an anomaly of HCP in comparison to HCP'. Consider another safe cyclic process (in the sense that it shares multiple sessions across two parallel processes), that is typable under the exact rules of [KMP19]:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{a_2}() \vdash_{\mathtt{H}} a_2 : \mathbf{1}}{a_1().\overline{a_2}() \vdash_{\mathtt{H}} a_1 : \bot \mid a_2 : \mathbf{1}}}{\overline{y}().a_1().\overline{a_2}() \vdash_{\mathtt{H}} a_1 : \bot \mid a_2 : \mathbf{1} \mid y : \mathbf{1}}}{x().\overline{y}().a_1().\overline{a_2}() \vdash_{\mathtt{H}} a_1 : \bot, x : \bot \mid a_2 : \mathbf{1} \mid y : \mathbf{1}}}{\overline{a_2}(y).x().\overline{y}().a_1().\overline{a_2}() \vdash_{\mathtt{H}} a_1 : \bot, x : \bot \mid a_2 : \mathbf{1} \otimes \mathbf{1}}}{a_1(x).\overline{a_2}(y).x().\overline{y}().a_1().\overline{a_2}() \vdash_{\mathtt{H}} a_1 : \bot \parr \bot \mid a_2 : \mathbf{1} \otimes \mathbf{1}} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{b_1}() \vdash_{\mathtt{H}} b_1 : \mathbf{1}}{b_2().\overline{b_1}() \vdash_{\mathtt{H}} b_1 : \mathbf{1} \mid b_2 : \bot}}{z().b_2().\overline{b_1}() \vdash_{\mathtt{H}} b_1 : \mathbf{1} \mid b_2 : \bot, z : \bot}}{\overline{n}().z().b_2().\overline{b_1}() \vdash_{\mathtt{H}} b_1 : \mathbf{1} \mid n : \mathbf{1} \mid b_2 : \bot, z : \bot}}{b_2(z).\overline{n}().z().b_2().\overline{b_1}() \vdash_{\mathtt{H}} b_1 : \mathbf{1} \mid n : \mathbf{1} \mid b_2 : \bot \parr \bot}}{\overline{b_1}(n).b_2(z).\overline{n}().z().b_2().\overline{b_1}() \vdash_{\mathtt{H}} b_1 : \mathbf{1} \otimes \mathbf{1} \mid b_2 : \bot \parr \bot}}{a_1(x).\overline{a_2}(y).x().\overline{y}().a_1().\overline{a_2}() \mid \overline{b_1}(n).b_2(z).\overline{n}().z().b_2().\overline{b_1}() \vdash_{\mathtt{H}} a_1 : \bot \parr \bot \mid a_2 : \mathbf{1} \otimes \mathbf{1} \mid b_1 : \mathbf{1} \otimes \mathbf{1} \mid b_2 : \bot \parr \bot}}{(\nu a_2 b_2)(a_1(x).\overline{a_2}(y).x().\overline{y}().a_1().\overline{a_2}() \mid \overline{b_1}(n).b_2(z).\overline{n}().z().b_2().\overline{b_1}()) \vdash_{\mathtt{H}} a_1 : \bot \parr \bot \mid b_1 : \mathbf{1} \otimes \mathbf{1}}}{(\nu a_1 b_1)(\nu a_2 b_2)(a_1(x).\overline{a_2}(y).x().\overline{y}().a_1().\overline{a_2}() \mid \overline{b_1}(n).b_2(z).\overline{n}().z().b_2().\overline{b_1}())) \vdash_{\mathtt{H}} \varnothing)}$$

It seems that HCP allows some cyclic processes. It would be worth exploring exactly which safe cyclic processes can be typed by HCP.

6.1 Future Work

Finishing the comparison In terms of future work, we seek to complete our project. In particular:

- We wish to prove our Operational Correspondence Conjectures (Conjectures 5.3.1, 5.3.2)
- Find if a unification result exists between $\mathcal{K}$ and $\mathcal{H}^\bullet$. That is, find if there is a type preserving translation from $\mathcal{K}$ to $\mathcal{H}^\bullet$, that has an operational correspondence property.

Further properties of $\mathcal{H}^\circ$ Furthermore, note how we separated $\mathcal{H}$ in terms of $\mathcal{K}$. This directly leads to separation results, which is why it was the approach chosen. However, we believe this set also captures two other properties.

The first is that all processes in $\mathcal{H}$ that would utilise non-blocking semantics is inside this set. Formally speaking, this would involve proving that for all $P \in \mathcal{H}^\circ$, we have that:

$$P \xrightarrow{l_1} \cdots \xrightarrow{l_n} \pi.Q \xrightarrow{\pi \| l} Q'$$

where the last transitions occurs due to RH-SelfSyn.

That is to say, that any process in $\mathcal{H}^\circ$ requires utilising non-blocking semantics at some point in its transitions. We leave such technical work as future work.

The second is that $\mathcal{H}^\circ$ is the set of all well-typed HCP-processes that do not have session progress. That is $\mathcal{H}^\circ = \mathcal{H} \cap \mathbb{D}$, where $\mathbb{D}$ is the set of all session processes that have session progress.

In the forward direction, any action that relies on a non-blocking interpretation must be blocking its continuation in a blocking interpretation, and so all its processes must deadlock. In the reverse direction, the rules that result in deadlocking behaviour must be those that are unique to HCP and these are the rules that involve non-blocking semantics, which must mean that a process that deadlocks (but is typable under HCP) utilises such rules, and therefore, must fall into $\mathcal{H}^\circ$, if the previous property holds true.

Extending HCP with priorities CP has already been extended to a type system that includes priorities, as explored by Dardha and Gay in [DG18]. However, it would be worth exploring the effects of extending HCP with priorities, as HCP captures novel behaviours not present in CP (via its non-blocking / self-synchronisation semantics).

7 Related Work

Curry-Howard Interpretations of Processes [Wad12] and [KMP19] are merely two instances of a Curry-Howard interpretation of processes, a project initiated by Abramsky in [Abr94] and Bellin and Scott in [BS94]. This project seeks to find a logical foundation for concurrency, and the last two papers sought to use the linear logic of Girard [Gir87] for this purpose. In the last decade or so, there has been a revitalisation in this line of work by Caires and Pfenning in [CP10], which presented a Curry-Howard Correspondence with intuitionistic linear logic and session processes. Both these systems capture deadlock-free processes, but there are other deadlock-free processes it does not capture, notably that which contain so-called "*safe*" cyclic process dependencies. The type system by Dardha and Gay in [DG18] captures exactly these processes. It does so by annotating linear logic types with *priorities*, which is a simplification of the obligations / capabilities notion in [Kob06], á la the type system by Padovani [Pad14].

More recently, Qian et al. introduced Client-Server Linear Logic [QKB21], which is an extension of linear logic, and a corresponding π-calculus for it. Another extension to linear logic, by Rocha and Caires in [RC21], adds the notion of shared state to CP, based on a "second-order Classical Linear Logic with mix". All these type systems thus far connect linear logic propositions with *binary* session types. That is, session types between two parties. [CMS23], in contrast, connects linear logic to *multiparty* session types, which are protocols between two or more participants. Lastly, we have [VDHP21], which extends PCP to support notions of asynchronous communication;

Comparisons of type systems The line of research involving comparing type systems of the π-calculus is relatively novel and to our knowledge, no new comparisons (at this level of rigour) have been formulated since [DP22]. The only similar comparison we have found is by van den Heuvel and Pérez in [HP20], which compares a type system derived from intuitionistic linear logic and from classical linear logic. It finds that the type system for classical linear logic is more expressive, in the sense that it captures more processes. One notable difference between it and [DP22] (as well as this paper) is that it defines type systems upon a single π-calculus, whereas [DP22] (and consequently this work) defines multiple π-calcululi.

8 Conclusion

We have extended a formal comparison of two type systems for the π-calculus that capture deadlock-free processes. We extend an existing comparison between [Kob06] and [Wad12] (which uses processes from [Vas12] as the unit of comparison), with a conceptual extension of [Wad12]: [KMP19].

One major difficulty for this is the presence of non-trivial differences in the operational semantics of [KMP19], in comparison to [Kob06], [Wad12], [Vas12]. Specifically, [KMP19] introduces notions of non-blocking actions and self-synchronisation, originally introduced by [MS04], that makes utilizing the methods of [DP22] inadequete. Specifically, replicating their methodology would result in a comparing a set of processes that may deadlock $\mathcal{H}$ to a set of processes that are deadlock-free ($\mathcal{L}, \mathcal{K}$ from [DP22]).

To resolve this, we first characterise the portion of $\mathcal{H}$ that lies outside $\mathcal{K}$: $\mathcal{H}^{\circ}$, by means of the predicate $\mathrm{NBR}(\cdot)$. Then, we provide a type-preserving translation $\mathcal{F}.(\cdot)$ that takes such processes and places them into the portion of $\mathcal{H}$ that lies inside $\mathcal{K}$: $\mathcal{H}^{\bullet}$.

We also considered how $\mathcal{H}^{\bullet}$ compares to $\mathcal{L}$ and $\mathcal{K}$. As expected, we found that $\mathcal{H}^{\bullet} \subsetneq \mathcal{K}$. Surprisingly, we found that $\mathcal{L} \not\subseteq \mathcal{H}^{\bullet}$ but rather, they simply intersected. In essence, adding mix principles to Wadler's Classical Processes produces syntactic structures not captured by HCP.

We also conjecture an operational correspondence property of that translation. If shown to be true, this would suggest that the non-blocking and self-synchronization semantics added are superfluous in terms of behaviour, and would make comparing $\mathcal{H}^{\bullet}$ to $\mathcal{L}, \mathcal{K}$ a fair characterisation for comparing [KMP19] to [Wad12],[Kob06].

To motivate the conjecture, we have provided representative examples of the transformation.

Acknowledgements We would like to acknowledge Prof. Dr. Jorge Peŕez, for supervising this project and providing invaluable insight and discussion during the process of writing this paper. We would also like to acknowledge Dr. Revantha Ramanayake for his additional supervision as well.

This work was completed as part of the Bachelor's Project course at the University of Groningen.

References

[Abr94] Samson Abramsky. Proofs as processes. *Theoretical Computer Science*, 135(1):5–9, December 1994.

[BS94] Gianluigi Bellin and Phillip J. Scott. On the π-calculus and linear logic. *Theoretical Computer Science*, 135(1):11–65, December 1994.

[CLM$^+$16] Marco Carbone, Sam Lindley, Fabrizio Montesi, Carsten Schürmann, and Philip Wadler. Coherence Generalises Duality: A Logical Explanation of Multiparty Session Types. In *International Conference on Concurrency Theory*, page 15 pages, 2016. Artwork Size: 15 pages Medium: application/pdf Publisher: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany.

[CMS23] Marco Carbone, Sonia Marin, and Carsten Schürmann. A logical interpretation of asynchronous multiparty compatibility, 2023.

[CP10] Luís Caires and Frank Pfenning. Session Types as Intuitionistic Linear Propositions. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 - Concurrency Theory*, Lecture Notes in Computer Science, pages 222–236, Berlin, Heidelberg, 2010. Springer.

[DG18] 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*, Lecture Notes in Computer Science, pages 91–109, Cham, 2018. Springer International Publishing.

[DGS12] Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session types revisited. In *Proceedings of the 14th Symposium on Principles and Practice of Declarative Programming*, PPDP '12, page 139–150, New York, NY, USA, 2012. Association for Computing Machinery.

[DP22] Ornela Dardha and Jorge A. Pérez. Comparing type systems for deadlock freedom. *Journal of Logical and Algebraic Methods in Programming*, 124:100717, 2022.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, January 1987.

[Gor10] Daniele Gorla. Towards a unified approach to encodability and separation results for process calculi. *Information and Computation*, 208(9):1031–1053, September 2010.

[HP20] Bas van den Heuvel and Jorge A. Pérez. Session Type Systems based on Linear Logic: Classical versus Intuitionistic. *Electronic Proceedings in Theoretical Computer Science*, 314:1–11, April 2020. arXiv:2004.01320 [cs].

[IK04] Atsushi Igarashi and Naoki Kobayashi. A generic type system for the Pi-calculus. *Theoretical Computer Science*, 311(1):121–163, January 2004.

[KL17] Naoki Kobayashi and Cosimo Laneve. Deadlock analysis of unbounded process networks. *Information and Computation*, 252:48–70, February 2017.

[KMP19] 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):24:1–24:29, January 2019.

[Kob02] Naoki Kobayashi. A Type System for Lock-Free Processes. *Information and Computation*, 177(2):122–159, September 2002.

[Kob06] Naoki Kobayashi. A New Type System for Deadlock-Free Processes. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, Christel Baier, and Holger Hermanns, editors, *CONCUR 2006 – Concurrency Theory*, volume 4137, pages 233–247. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. Series Title: Lecture Notes in Computer Science.

[Kob07] Naoki Kobayashi. Type systems for concurrent programs, extended version of [?], Tohoku University, www.kb.ecei.tohoku.ac.jp/koba/papers/tutorial- type- extended.pdf, 2007.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Information and Computation*, 100(1):1–40, September 1992.

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

[Pad14] 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, pages 1–10, New York, NY, USA, July 2014. Association for Computing Machinery.

[QKB21] Zesen Qian, G.A. Kavvos, and Lars Birkedal. Client-Server Sessions in Linear Logic, March 2021. arXiv:2010.13926 [cs].

[RC21] Pedro Rocha and Luís Caires. Propositions-as-types and shared state. *Proceedings of the ACM on Programming Languages*, 5(ICFP):1–30, August 2021.

[TVTV13] Hugo Torres Vieira and Vasco Thudichum Vasconcelos. Typing Progress in Communication-Centred Systems. In Rocco De Nicola and Christine Julien, editors, *Coordination Models and Languages*, Lecture Notes in Computer Science, pages 236–250, Berlin, Heidelberg, 2013. Springer.

[Vas12] Vasco T. Vasconcelos. Fundamentals of session types. *Information and Computation*, 217:52–70, August 2012.

[VDHP21] Bas Van Den Heuvel and Jorge A. Pérez. Deadlock Freedom for Asynchronous and Cyclic Process Networks. In *Electronic Proceedings in Theoretical Computer Science*, volume 347, pages 38–56, October 2021. ISSN: 2075-2180 Journal Abbreviation: Electron. Proc. Theor. Comput. Sci.

[Wad12] Philip Wadler. Propositions as sessions. In *Proceedings of the 17th ACM SIGPLAN international conference on Functional programming*, ICFP '12, pages 273–286, New York, NY, USA, September 2012. Association for Computing Machinery.

[Yos96] Nobuko Yoshida. Graph types for monadic mobile processes. In Gerhard Goos, Juris Hartmanis, Jan Leeuwen, V. Chandru, and V. Vinay, editors, *Foundations of Software Technology and Theoretical Computer Science*, volume 1180, pages 371–386. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996. Series Title: Lecture Notes in Computer Science.

A Omitted proofs for §4

A.1 Proof of theorem 4.2.1

We divide the proof into two lemmas: Lemma A.1.1 and Lemma A.1.2.

Lemma A.1.1. $P \in \mathcal{H}^o \implies \text{NBR}(P)$

Proof. Suppose $P \in \mathcal{H}^o$. We will prove NBR(P) by showing that $\neg$NBR(P) leads to a contradiction.

Suppose $\neg$NBR(P). We will show that $P \in \mathcal{K}$ via structural induction.

1. Consider $P = \mathbf{0}$. Let $\Gamma = \Gamma_1, \ldots, \Gamma_k$. The proof follows as in Lemma A.2, case 1 in [DP22], except we have

$$\mathbf{0} \vdash_{\mathtt{H}} x_{1,1} : \bullet, \ldots, x_{1,n_1} : \bullet \mid \cdots \mid x_{k,1} : \bullet, \ldots, x_{k,n_k} : \bullet$$

for some $x_{i,j} \in \text{cn}(\Gamma_i)$ and $n_i = \text{cn}(\Gamma_i)$.
instead of

$$\mathbf{0} \vdash_{\mathtt{LL}} x_1 : \bullet, \ldots, x_n : \bullet$$

but the subsequent justification remains the same, as $\Gamma^{\downarrow} = \emptyset$ and we use $\vdash_{\prec}$ instead of $\vdash^{\mu}_{\prec}$.

2. Consider $P = x(y).P'$. Let $\Gamma = \Gamma_1, \ldots, \Gamma_k$. The proof follows as in Lemma A.2, case 2 in [DP22], except we use definitions of $\mathcal{H}$ instead of $\mathcal{L}$ and $\vdash_{\prec}$ instead of $\vdash^{\mu}_{\prec}$.

3. Consider $P = \overline{x}\langle y\rangle.P'$. We have:

$$\Gamma_1, \ldots, \Gamma_k, x : !T.S, y : T \vdash_{\mathtt{ST}} \overline{x}\langle y\rangle.P'$$

for some contexts $\Gamma_1, \ldots, \Gamma_k$ and session types S, T.
Let $\Gamma = \Gamma_1, \ldots, \Gamma_k$.
By inversion on typing judgment on the above, we have:

$$\Gamma, x : S \vdash_{\mathtt{ST}} P'$$

By induction hypothesis on the above, we have:

$$[\![\Gamma^{\downarrow}]\!]^{f'}_{\mathrm{u}}, f'_x : [\![S]\!]_{\mathrm{u}} \vdash_{\prec'} [\![P']\!]^{f'}_{\mathrm{u}} \tag{1}$$

for some renaming function f' and some $\prec$ such that $\prec' = \prec \cup \{(c, y) \mid y \in \text{fn}([\![P']\!]^{f,\{x \mapsto c\}}_{\mathrm{u}}) \setminus \{c\}\}$.
Now we must show:

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

We let f be such that $f' = f, \{x \mapsto c\}$. We can rewrite (1) as follows:

$$[\![\Gamma^{\downarrow}]\!]^{f}_{\mathrm{u}}, c : [\![S]\!]_{\mathrm{u}} \vdash_{\prec'} [\![P']\!]^{f,\{x \mapsto c\}}_{\mathrm{u}} \tag{2}$$

Applying Tπ-Var and Tπ-Tup, we can derive:

$$y : [\![T]\!]_{\mathrm{u}}, c : [\![\overline{S}]\!]_{\mathrm{u}} \vdash_{\prec'} y : [\![T]\!]_{\mathrm{u}}, c : [\![\overline{S}]\!]_{\mathrm{u}} \tag{3}$$

Applying Tπ-Out to (2) and (3) we derive:

$$f_x : \mathrm{chan}(\llbracket T \rrbracket_{\mathrm{u}}, \llbracket \overline{S} \rrbracket_{\mathrm{u}}; !^0_0);_{\prec'} (c : \llbracket S \rrbracket_{\mathrm{u}} \mid \llbracket \overline{S} \rrbracket_{\mathrm{u}}, y : \llbracket T \rrbracket_{\mathrm{u}} \mid \llbracket \Gamma^{\downarrow} \rrbracket_{\mathrm{u}}) \vdash_{\prec'} \overline{f_x}\langle y, c\rangle.\llbracket P' \rrbracket^{f,\{x \mapsto c\}}_{\mathrm{u}} \tag{4}$$

Let $U_1 = u(\llbracket S \rrbracket_{\mathrm{u}})$, $U_2 = u(\llbracket \overline{S} \rrbracket_{\mathrm{u}})$. Observe that $\mathrm{rel}(U_1 \mid U_2)$

Observing that $\llbracket S \rrbracket_{\mathrm{u}} \mid \llbracket \overline{S} \rrbracket_{\mathrm{u}} = \mathrm{chan}(\tilde{\tau}; U_1 \mid U_2)$ and applying Tπ-Res to (4), we have:

$$f_x : \mathrm{chan}(\llbracket T \rrbracket_{\mathrm{u}}, \llbracket \overline{S} \rrbracket_{\mathrm{u}}; !^0_0);_{\prec} (y : \llbracket T \rrbracket_{\mathrm{u}} \mid \llbracket \Gamma^{\downarrow} \rrbracket_{\mathrm{u}}) \vdash_{\prec} (\nu c)\overline{f_x}\langle y, c\rangle.\llbracket P' \rrbracket^{f,\{x \mapsto c\}}_{\mathrm{u}}$$

which proves our thesis.

4. Consider $P = x \triangleright \{l_i : P_i\}_{i \in I}$. Let $\Gamma = \Gamma_1, \ldots, \Gamma_k$. The proof follows as in Lemma A.2, case 4 in [DP22], except we use definitions of $\mathcal{H}$ instead of $\mathcal{L}$ and $\vdash_{\prec}$ instead of $\vdash^{\mu}_{\prec}$.

5. Consider $P = x \triangleleft l_j.P_j$. Let $\Gamma = \Gamma_1, \ldots, \Gamma_k$. The proof follows as in Lemma A.2, case 5 in [DP22], except we use definitions of $\mathcal{H}$ instead of $\mathcal{L}$ and $\vdash_{\prec}$ instead of $\vdash^{\mu}_{\prec}$.

6. Consider $P = (\nu xy)P'$. Observe that we have:

$$\Gamma_1, \ldots, \Gamma_k \vdash_{\mathtt{ST}} (\nu xy)(P') \tag{5}$$

$$(\nu xy)\llbracket P' \rrbracket_h \vdash_{\mathtt{H}} \llbracket \Gamma_1 \rrbracket_h \mid \cdots \mid \llbracket \Gamma_k \rrbracket_h \tag{6}$$

By inversion on typing judgements on (6) we have:

$$\frac{\llbracket P' \rrbracket_h \vdash_{\mathtt{H}} \llbracket \Gamma_1 \rrbracket_h \mid \cdots \mid \llbracket \Gamma_i \rrbracket_h, x : \llbracket T \rrbracket_h \mid \cdots \mid \llbracket \Gamma_j \rrbracket_h, y : \llbracket \overline{T} \rrbracket_h \mid \cdots \mid \llbracket \Gamma_k \rrbracket_h}{(\nu xy)\llbracket P' \rrbracket_h \vdash_{\mathtt{H}} \llbracket \Gamma_1 \rrbracket_h \mid \cdots \mid \llbracket \Gamma_k \rrbracket_h} \tag{7}$$

For $\llbracket T \rrbracket_h$ to be assigned to x (or y), either:

- $\llbracket P' \rrbracket_h = D[P'_1.\pi.P''] = D'[P'_1.\pi.P'' \mid P_2]$
- $\llbracket P' \rrbracket_h = D[P'_1.x \triangleright \{l_i : P_i\}_{i \in I}] = D'[P'_1.x \triangleright \{l_i : P_i\}_{i \in I} \mid P_2]$
- $\llbracket P' \rrbracket_h = D[P'_1.[x \leftrightarrow z]] = D'[P'_1.[x \leftrightarrow z] \mid P_2]$

Where $\mathrm{sub}(\pi) = x$ (or $\mathrm{sub}(\pi) = y$).

In all cases, by $\neg\mathrm{NBR}(P)$, we must $y \notin \mathrm{fn}(P''), y \notin \mathrm{fn}(P'_1), y \notin \mathrm{fn}(P_i)$. Thus, $y \in \mathrm{fn}(P_2)$, as $y \in \mathrm{fn}(P')$. This is analogous for y.

Let $P_1 = P'_1.\pi.P'', P_1 = P'_1.x \triangleright \{l_i : P_i\}_{i \in I}, P_1 = P'_1.[x \leftrightarrow z]$ in the respective cases. Then continuing the inversion on typing judgements in (7):

$$\dfrac{\dfrac{\dfrac{\llbracket P_1 \rrbracket_h \vdash_{\mathtt{H}} \mathcal{G}_1 \mid \llbracket \Delta_1 \rrbracket_h, x : \llbracket S \rrbracket_h \qquad \llbracket P_2 \rrbracket_h \vdash_{\mathtt{H}} \mathcal{G}_2 \mid \llbracket \Delta_2 \rrbracket_h, y : \llbracket S' \rrbracket_h}{\vdots}\,\text{H-Mix} \qquad \vdots}{\llbracket P' \rrbracket_h \vdash_{\mathtt{H}} \llbracket \Gamma_1 \rrbracket_h \mid \cdots \mid \llbracket \Gamma_i \rrbracket_h, x : \llbracket T \rrbracket_h \mid \cdots \mid \llbracket \Gamma_j \rrbracket_h, y : \llbracket \overline{T} \rrbracket_h \mid \cdots \mid \llbracket \Gamma_k \rrbracket_h}}{(\nu xy)\llbracket P' \rrbracket_h \vdash_{\mathtt{H}} \llbracket \Gamma_1 \rrbracket_h \mid \cdots \mid \llbracket \Gamma_k \rrbracket_h}$$

Then observe that $S = T$ and $S' = \overline{T}$, via $\neg\mathrm{NBR}(P)$. If $S \neq T$, we must have $P' = C_2[\pi.C_1[P_1 \mid P_2]]$, with $\mathrm{sub}(\pi) = x \vee \mathrm{sub}(\pi) = y$, but this is disallowed under $\neg\mathrm{NBR}(P)$. A similar argument follows for S'.

Thus, under our inductive hypothesis $\llbracket P_1 \rrbracket^{f',\{x,y \mapsto c\}}_{\mathrm{u}}$ is typable under a context with type assignment $c : \llbracket T \rrbracket_{\mathrm{u}}$ and $\llbracket P_2 \rrbracket^{f',\{x,y \mapsto c\}}_{\mathrm{u}}$ is typable under a context with type assignment $c : \llbracket \overline{T} \rrbracket_{\mathrm{u}}$. Then $\llbracket P_1 \mid P_2 \rrbracket^{f',\{x,y \mapsto c\}}_{\mathrm{u}}$ has a context with type assignment $c : \llbracket T \rrbracket_{\mathrm{u}} \mid \llbracket \overline{T} \rrbracket_{\mathrm{u}}$.

So we have $\Gamma, c : [\![T]\!]_{\mathrm{u}} \mid [\![\overline{T}]\!]_{\mathrm{u}} \vdash_{\prec} [\![P_1 \mid P_2]\!]_{\mathrm{u}}^{f', \{x,y \mapsto c\}}$. Now observe that $P' \neq D_1[\pi.D_1[P_1 \mid P_2]]$ where $\mathrm{sub}(\pi) \in \{x, y\}$, as this would contradict $\mathrm{NBR}(P)$. Thus we will have

$$\Gamma', c : [\![T]\!]_{\mathrm{u}} \mid [\![\overline{T}]\!]_{\mathrm{u}} \vdash_{\prec} [\![P']\!]_{\mathrm{u}}^{f^*, \{x,y \mapsto c\}}$$

Let $U = u([\![T]\!]_{\mathrm{u}} \mid [\![\overline{T}]\!]_{\mathrm{u}})$. We have it that $\mathrm{rel}(U)$, via Corollary 2.4.1. As as $\mathrm{rel}(U)$, we can apply Tπ-Res over channel c, and we have our thesis. Thus, we have:

$$\frac{\Gamma', c : [\![T]\!]_{\mathrm{u}} \mid [\![\overline{T}]\!]_{\mathrm{u}} \vdash_{\prec} [\![P']\!]_{\mathrm{u}}^{f^*, \{x,y \mapsto c\}} \qquad \mathrm{rel}(U)}{\Gamma' \vdash_{\prec} (\nu c)[\![P']\!]_{\mathrm{u}}^{f^*, \{x,y \mapsto c\}}}$$

And we conclude this case.

7. Consider $P = P_1 \mid P_2$. Let $\Gamma = \Gamma_1, \ldots, \Gamma_k$. The proof follows as in Lemma A.2, case 6 in [DP22], except we use definitions of $\mathcal{H}$ instead of $\mathcal{L}$ and $\vdash_{\prec}$ instead of $\vdash^{\mu}_{\prec}$. The proof holds as because $P \in \mathcal{H}$, every channel name in Γ must be unique and hence splitting it must have the split domains share no names.

As $P \in \mathcal{K}$, then $P \notin \mathcal{H}^o$. However, this is a contradiction and so, $\neg\neg\mathrm{NBR}(P) \equiv \mathrm{NBR}(P)$. Thus, we have our thesis. □

Lemma A.1.2. $(P \in \mathcal{H} \wedge \mathrm{NBR}(P)) \implies P \in \mathcal{H}^o$

Proof. Suppose that $\mathrm{NBR}(P)$. Then $\exists D, C, P' : P = D[(\nu xy)C[\pi.P']]$, where $\mathrm{sub}(\pi) = s \in \{x, y\}$ and $o \in \mathrm{fn}(P')$ where $o \in \{x, y\} \setminus \{s\}$.

Observe that $\exists \Gamma : \Gamma \vdash_{\prec} [\![P]\!]_{\mathrm{u}}^{f} \implies \exists \Gamma' : \Gamma' \vdash_{\prec'} [\![(\nu xy)C[\pi.P']]\!]_{\mathrm{u}}^{f'}$ (as all well-typed processes must have well-typed subprocesses).

We will show that $\neg\exists\Gamma' : \Gamma' \vdash_{\prec'} [\![C[\pi.P']]\!]_{\mathrm{u}}^{f'}$ and thus, by contrapositive, we have $\neg\exists\Gamma : \Gamma \vdash_{\prec} [\![P]\!]_{\mathrm{u}}^{f}$.

Suppose $\exists\Gamma' : \Gamma' \vdash_{\prec'} [\![(\nu xy)C[\pi.P']]\!]_{\mathrm{u}}^{f'}$. By inversion on typing judgements:

$$\frac{\Gamma', c : \mathrm{chan}(\widetilde{\tau}; U) \vdash_{\prec' \cup \{(c,y) \mid y \in \mathrm{fn}(P) \setminus \{c\}\}} [\![C[\pi.P']]\!]_{\mathrm{u}}^{f', \{x,y \mapsto c\}} \qquad \mathrm{rel}(U)}{\Gamma' \vdash_{\prec'} [\![(\nu xy)C[\pi.P']]\!]_{\mathrm{u}}^{f'}}$$

Thus, if we show $\neg\mathrm{rel}(U)$, we will have a contradiction.

To show that $\neg\mathrm{rel}(U)$, it suffices to show that $U = \alpha.U'$, where α is a sequential usage. We will show this by applying Lemma A.1.3 to $[\![C[\pi.P']]\!]_{\mathrm{u}}^{f', \{x,y \mapsto c\}}$.

We can apply this Lemma as we satisfy its conditions:

- We have $C[\pi.P'] \in \mathcal{H}$ as $(\nu xy)C[\pi.P'] \in \mathcal{H}$ (via Lemma A.1.4) and $(\nu xy)C[\pi.P'] \in \mathcal{H} \implies C[\pi.P'] \in \mathcal{H}$, via inversion of typing judgements on $(\nu xy)C[\pi.P']$.
- The rest of the conditions are immediate from the definition of $\mathrm{NBR}(\cdot)$.

Therefore, we have a contradiction and $\neg\exists\Gamma' : \Gamma' \vdash_{\prec'} [\![(\nu xy)P']\!]_{\mathrm{u}}^{f'}$.

Therefore, through the aforementioned contrapositive, we have $\neg\exists\Gamma : \Gamma \vdash_{\prec} [\![P]\!]_{\mathrm{u}}^{f}$.

Thus $P \notin \mathcal{K}$ and as $P \in \mathcal{H}$, we have $P \in \mathcal{H}^o$. □

Lemma A.1.3. *Consider $C_1[\pi.P] \in \mathcal{H}$, and x, y, π such that:*

- $\mathrm{sub}(\pi) = s \wedge s \in \{x, y\}$
- $o \in \{x, y\} \setminus \{s\} \wedge o \in \mathrm{fn}(P)$
- $x, y \in \mathrm{fn}(C_1[\pi.P])$

Then:

$$\Gamma, c : \mathrm{chan}(\widetilde{\tau}; U) \vdash_{\prec} [\![C_1[\pi.P]]\!]_{\mathrm{u}}^{f,\{x,y \mapsto c\}} \implies U = \alpha.U'$$

Proof. We prove this via structural induction on the definition of the process context C_1, on the process $[\![C_1[\pi.P']]\!]_{\mathrm{u}}^{f,\{x,y \mapsto c\}}$.

1. Consider $C_1[\pi.P] = \overline{v}\langle w\rangle.C'[\pi.P]$

   Suppose:
   $$\Gamma, c : \mathrm{chan}(\widetilde{\tau}; U) \vdash_{\prec} [\![\overline{v}\langle w\rangle.C'[\pi.P]]\!]_{\mathrm{u}}^{f,\{x,y \mapsto c\}}$$
   Let $f' = f, \{x, y \mapsto c\}$.
   Then
   $$\Gamma, c : \mathrm{chan}(\widetilde{\tau}; U) \vdash_{\prec} (\nu d)\overline{f'_v}\langle f'_w, d\rangle.[\![C'[\pi.P]]\!]_{\mathrm{u}}^{f',\{v \mapsto d\}}$$

   Then by inversion on typing judgments:
   $$\cfrac{\cfrac{\Gamma_1 \vdash_{\prec'} [\![C'[\pi.P]]\!]_{\mathrm{u}}^{f',\{v \mapsto d\}} \qquad \Gamma_2 \vdash_{\prec'} f'_w, d : \widetilde{\tau'}}{\Gamma, c : \mathrm{chan}(\widetilde{\tau}; U), d : \mathrm{chan}(\widetilde{\tau'}; U_1) \vdash_{\prec'} \overline{f'_v}\langle f'_w, d\rangle.[\![C'[\pi.P]]\!]_{\mathrm{u}}^{f',\{v \mapsto d\}}} \quad \mathrm{rel}(U_1)}{\Gamma, c : \mathrm{chan}(\widetilde{\tau}; U) \vdash_{\prec} (\nu d)\overline{f'_v}\langle f'_w, d\rangle.[\![C'[\pi.P]]\!]_{\mathrm{u}}^{f',\{v \mapsto d\}}}$$

   With $\Gamma, c : \mathrm{chan}(\widetilde{\tau}; U), d : \mathrm{chan}(\widetilde{\tau'}; U_1) = f'_v : \mathrm{chan}(\widetilde{\tau}; !^0_\kappa);_{\prec} (\Gamma_1 \mid \Gamma_2)$.

   There are two cases:

   - Consider $v \in \{x, y\}$. We have $f'_v = c$. Then
     $$\Gamma, c : \mathrm{chan}(\widetilde{\tau}; U), d : \mathrm{chan}(\widetilde{\tau'}; U_1) = c : \mathrm{chan}(\widetilde{\tau}; !^0_\kappa);_{\prec} (\Gamma_1 \mid \Gamma_2)$$
     Observe then that $U = !^0_\kappa.U'$, for some U'. Thus, we have achieved our thesis.
   - Consider $v \notin \{x, y\}$. Then by our induction hypothesis, $\Gamma_1 = \Gamma'_1, c : \mathrm{chan}(\widetilde{\tau^*}; U^*)$, with $U^* = \alpha.U'$. Observing that $c \notin \mathrm{dom}(\Gamma_2)$, and $f'_v \neq c$, $U = U^* = \alpha.U'$, and thus, we have achieved our thesis.

2. Consider $C_1[\pi.P] = v \triangleleft l_j.C'[\pi.P]$

   This case is similar to the previous case.

3. Consider $C_1[\pi.P] = v(w).C'[\pi.P]$

   Suppose:
   $$\Gamma, c : \mathrm{chan}(\widetilde{\tau}; U) \vdash_{\prec} [\![v(w).C'[\pi.P]]\!]_{\mathrm{u}}^{f,\{x,y \mapsto c\}}$$
   Let $f' = f, \{x, y \mapsto c\}$.
   Then
   $$\Gamma, c : \mathrm{chan}(\widetilde{\tau}; U) \vdash_{\prec} f'_v(w, d).[\![C'[\pi.P]]\!]_{\mathrm{u}}^{f',\{v \mapsto d\}}$$

Then by inversion on typing judgments:

$$\frac{\Gamma' \vdash_{\prec} [\![C'[\pi.P]]\!]_{\mathrm{u}}^{f',\{v\mapsto d\}}}{\Gamma, c : \mathrm{chan}(\widetilde{\tau}; U) \vdash_{\prec} f'_v(w,d).[\![C'[\pi.P]]\!]_{\mathrm{u}}^{f',\{v\mapsto d\}}}$$

There are two cases:

- Consider $v \in \{x, y\}$. Let $\Gamma' = \Delta, w, d : \widetilde{\tau^*}$. Observing that $f'_v = c$, we will have

  $$\Gamma, c : \mathrm{chan}(\widetilde{\tau}; U) = c : \mathrm{chan}(\widetilde{\tau^*}; ?^0_\kappa);_{\prec} \Delta$$

  Then $U = ?^0_\kappa.U'$, for some U' and so, we have our thesis.
- Consider $v \notin \{x, y\}$. Observe that as $x, y \in \mathrm{fn}(v(w).C'[\pi.P])$, then $x, y \in \mathrm{fn}(C'[\pi.P])$, meaning that $c \in \mathrm{dom}(\Gamma')$. Thus $\Gamma' = \Gamma'', c : \mathrm{chan}(\widetilde{\tau'}; U^*)$. By the induction hypothesis, $U^* = \alpha.U'$.
  Now let $\Gamma' = \Delta, c : \mathrm{chan}(\widetilde{\tau'}; U^*), w, d : \widetilde{\tau^*}$.
  We have

  $$\Gamma, c : \mathrm{chan}(\widetilde{\tau}; U) = f'_v : \mathrm{chan}(\widetilde{\tau^*}; ?^0_\kappa);_{\prec} (\Delta, c : \mathrm{chan}(\widetilde{\tau'}; U^*)) = c : \mathrm{chan}(\widetilde{\tau'}; U^*), f'_v : \mathrm{chan}(\widetilde{\tau^*}; ?^0_\kappa);_{\prec} \Delta$$

  Thus, $U = U^* = \alpha.U'$, and we have achieved our thesis.

4. Consider $C_1[\pi.P] = C'[\pi.P] \mid Q$.
   As $C_1[\pi.P] \in \mathcal{H}$, by inversion of typing judgments we have:

   $$\frac{[\![C'[\pi.P]]\!]_h \vdash_{\mathtt{H}} \mathcal{G}_1 \qquad [\![Q]\!]_h \vdash_{\mathtt{H}} \mathcal{G}_2}{[\![C'[\pi.P] \mid Q]\!]_h \vdash_{\mathtt{H}} \mathcal{G}_1 \mid \mathcal{G}_2}$$

   Suppose $\{x, y\} \cap \mathrm{cn}(\mathcal{G}_2) \neq \emptyset$. Then $\{x, y\} \cup \mathrm{cn}(\mathcal{G}_1) \neq \mathrm{cn}(\mathcal{G}_1)$, as otherwise $\mathcal{G}_1 \mid \mathcal{G}_2$ is not defined. That is to say, suppose $\mathcal{G}_2$ assigns a type to at least one of x, y.

   However, then $x \in \mathrm{fn}(Q) \vee y \in \mathrm{fn}(Q)$ and $x \notin \mathrm{fn}(C'[\pi.P]) \vee y \notin \mathrm{fn}(C'[\pi.P])$, meaning that the free occurrences of x, y in P are bound occurrences in $C'[\pi.P]$. However, then we have names in binding occurences in $C'[\pi.P] \mid Q$ that are not distinct from the free names in $C'[\pi.P] \mid Q$ and thus would not follow Barendregt's variable convention.

   Therefore, $x, y \notin \mathrm{cn}(\mathcal{G}_2)$ and $x, y \notin \mathrm{fn}(Q)$. However, $x, y \in \mathrm{fn}(C_1[\pi.P])$ and so, $x, y \in \mathrm{fn}(C'[\pi.P])$.

   Now, if
   $$\Gamma, c : \mathrm{chan}(\widetilde{\tau}; U) \vdash_{\prec} [\![C'[\pi.P] \mid Q]\!]_{\mathrm{u}}^{f,\{x,y\mapsto c\}}$$

   By inversion of typing judgments:

   $$\frac{\Gamma_1 \vdash_{\prec} [\![C'[\pi.P]]\!]_{\mathrm{u}}^{f,\{x,y\mapsto c\}} \qquad \Gamma_2 \vdash_{\prec} [\![Q]\!]_{\mathrm{u}}^{f,\{x,y\mapsto c\}}}{\Gamma, c : \mathrm{chan}(\widetilde{\tau}; U) \vdash_{\prec} [\![C'[\pi.P] \mid Q]\!]_{\mathrm{u}}^{f,\{x,y\mapsto c\}}}$$

   With $\Gamma, c : \mathrm{chan}(\widetilde{\tau}; U) = \Gamma_1 \mid \Gamma_2$.

   Observe that $c \notin \mathrm{cn}(\Gamma_2)$, as $x, y \notin \mathrm{fn}(Q)$. But as $c \in \mathrm{cn}(\Gamma_1 \mid \Gamma_2)$, $c \in \mathrm{cn}(\Gamma_1)$. Thus, $\Gamma_1 = \Gamma'_1, c : \mathrm{chan}(\widetilde{\tau'}; U^*)$.

   As $c \notin \mathrm{cn}(\Gamma_2)$, we have $\Gamma_1 \mid \Gamma_2 = c : \mathrm{chan}(\widetilde{\tau'}; U^*), (\Gamma'_1 \mid \Gamma_2)$.

   Thus, $\widetilde{\tau'} = \widetilde{\tau}$ and $U = U^*$. Now applying our inductive hypothesis to $C'[\pi.P]$, we have $U^* = \alpha.U'$

   And thus we have our thesis.

5. Consider $C_1[\pi.P] = (\nu vw)C'[\pi.P]$.

   If we have
$$\Gamma, c : \mathrm{chan}(\tilde{\tau}; U) \vdash_{\prec} [\![(\nu vw)C'[\pi.P]]\!]_{\mathrm{u}}^{f,\{x,y\mapsto c\}}$$

   By inversion on typing judgments:
$$\frac{\Gamma, c : \mathrm{chan}(\tilde{\tau}; U), d : \mathrm{chan}(\tilde{\tau'}; U') \vdash_{\prec'} [\![C'[\pi.P]]\!]_{\mathrm{u}}^{f,\{x,y\mapsto c\},\{v,w\mapsto d\}} \qquad \mathrm{rel}(U')}{\Gamma, c : \mathrm{chan}(\tilde{\tau}; U) \vdash_{\prec} [\![(\nu vw)C'[\pi.P]]\!]_{\mathrm{u}}^{f,\{x,y\mapsto c\}}}$$

   $v, w \notin \{x, y\}$, as then $x, y \notin \mathrm{fn}(C_1[\pi.P])$, which would violate our assumption $x, y \in \mathrm{fn}(C_1[\pi.P])$.

   Observe then that:
$$\Gamma', c : \mathrm{chan}(\tilde{\tau}; U) \vdash_{\prec'} [\![C'[\pi.P]]\!]_{\mathrm{u}}^{f',\{x,y\mapsto c\}}$$

   And by our inductive hypothesis we have $U = \alpha.U^*$

   Thus, we have our thesis.

6. Consider $C_1[\pi.P] = \pi.P$. Observe that as $\mathrm{sub}(\pi) = x \;\vee\; \mathrm{sub}(\pi) = y$, $\mathrm{sub}([\![\pi]\!]_{\mathrm{u}}^{f,\{x,y\mapsto c\}}) = c$. Thus, if we have
$$\Gamma, c : \mathrm{chan}(\tilde{\tau}; U) \vdash_{\prec} [\![\pi.P]\!]_{\mathrm{u}}^{f,\{x,y\mapsto c\}}$$

   $U = \alpha.U'$, where

   - $\alpha = !^{o}_{\kappa}$ if $\pi = \overline{x}\langle v\rangle \vee \pi = x \triangleleft l_j$
   - $\alpha = ?^{o}_{\kappa}$ if $\pi = x(w)$

   For some U', as π is the last action over channel c.

   Thus, we have our thesis.

□

Lemma A.1.4.
$$C[P] \in \mathcal{H} \implies P \in \mathcal{H}$$

Proof. We prove this via structural induction on C.

In the case: $C[P] = P$. Then our thesis follows immediately.

In other cases, we either have:

- $C[P] = \pi.C'[P]$
- $C[P] = C'[P] \mid Q$
- $C[P] = (\nu xy)C'[P]$

Thus, if $C[P] \in \mathcal{H}$, $C[P] \vdash_{\mathtt{H}} \mathcal{G}$, which implies $C'[P] \vdash_{\mathtt{H}} \mathcal{G}'$, for some $\mathcal{G}, \mathcal{G}'$. Observe then that $C'[P] \in \mathcal{H}$ and thus by our inductive hypothesis, $P \in \mathcal{H}$.

Thus, we have our thesis, via induction. □

B Omitted proofs for §5

As a reminder, various proofs of HCP typing judgements (such as $\triangledown_i, \triangledown', \triangledown^*$) are implicitly defined via Convention 3.1.1.

B.1 Proof of Lemma 5.3.1

We repeat Lemma 5.3.1

$$\llbracket P \rrbracket_h \vdash_{\mathtt{H}} \mathcal{G} \wedge P \notin \mathcal{K} \implies \llbracket \mathcal{F}'_{\triangledown}(P) \rrbracket_h \vdash_{\mathtt{H}} \mathcal{G}$$

for all proofs, $\triangledown$, of $\llbracket P \rrbracket_h \vdash_{\mathtt{H}} \mathcal{G}$.

Proof. We assume $\llbracket P \rrbracket_h \vdash_{\mathtt{H}} \mathcal{G} \wedge P \notin \mathcal{K}$. Let $\triangledown$ be any proof of $\llbracket P \rrbracket_h \vdash_{\mathtt{H}} \mathcal{G}$. We perform case analysis on the definition of $\mathcal{F}'_{\cdot}(\cdot)\cdot$.

1. $\mathcal{F}'_{\triangledown}(P) = D[(\nu xy)(\mathcal{R}^{N_1}_{\triangledown^*}(C[\pi.P']) \mid \mathcal{K}^{N_1}_{\triangledown^*}(C[\pi.P']))]$.

   Observe that that $P = D[(\nu xy)C[\pi.P']]$.

   Now by definition, $\triangledown$ must be:

$$\begin{array}{c} \vdots \\ \hline \llbracket C[\pi.P'] \rrbracket_h \vdash_{\mathtt{H}} \mathcal{G}'' \mid \Delta, x : T \mid \Theta, y : \overline{T} \triangleq \mathcal{G}^* \\ \hline \llbracket (\nu xy)C[\pi.P'] \rrbracket_h \vdash_{\mathtt{H}} \mathcal{G}' \\ \hline \vdots \\ \hline \llbracket P \rrbracket_h \vdash_{\mathtt{H}} \mathcal{G} \end{array}$$

   Observe that as $\llbracket C[\pi.P'] \rrbracket_h \vdash_{\mathtt{H}} \mathcal{G}^*$, then $\llbracket \mathcal{R}^{N_1}_{\triangledown^*}(C[\pi.P']) \mid \mathcal{K}^{N_1}_{\triangledown^*}(C[\pi.P'])\rrbracket_h \vdash_{\mathtt{H}} \mathcal{G}^*$, by Lemma B.1.1, as N_1 is the set of names in the context containing x, in $\mathcal{G}$, by definition.

   Thus, $\llbracket D[(\nu xy)(\mathcal{R}^{N_1}_{\mathcal{G}^*}(C[\pi.P']) \mid \mathcal{K}^{N_1}_{\mathcal{G}^*}(C[\pi.P']))] \rrbracket_h \vdash_{\mathtt{H}} \mathcal{G}$, by Lemma B.1.4.

   Thus, we conclude this case.

2. $\mathcal{F}'_{\triangledown}(P) = D[(\nu xy)(\mathcal{R}^{N_2}_{\triangledown^*}(C[\pi.P']) \mid \mathcal{K}^{N_2}_{\triangledown^*}(C[\pi.P']))]$. Analogous to case 1.

3. $\mathcal{F}'_{\triangledown}(P) = P$. Follows immediately.

□

Lemma B.1.1.

$$\llbracket P \rrbracket_h \vdash_{H} \mathcal{G} \implies \llbracket \mathcal{R}^{\mathrm{cn}(\Gamma_i)}_{\triangledown}(P) \mid \mathcal{K}^{\mathrm{cn}(\Gamma_i)}_{\triangledown}(P) \rrbracket_h \vdash_{H} \mathcal{G}$$

where $\mathcal{G} = \llbracket \Gamma_1 \rrbracket_h \mid \cdots \mid \llbracket \Gamma_n \rrbracket_h$ and $\triangledown$ is any proof of $\llbracket P \rrbracket_h \vdash_{\mathtt{H}} \mathcal{G}$.

Proof. Suppose

$$\llbracket P \rrbracket_h \vdash_{\mathtt{H}} \mathcal{G}$$

where $\mathcal{G} = \llbracket \Gamma_1 \rrbracket_h \mid \cdots \mid \llbracket \Gamma_n \rrbracket_h$ and $\triangledown$ is any proof of $\llbracket P \rrbracket_h \vdash_{\mathtt{H}} \mathcal{G}$.

Observe by Lemma B.1.2, we have:

$$[\![\mathcal{R}^{\mathrm{cn}(\Gamma_i)}_{\triangledown}(P)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{i+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$

Observe by Lemma B.1.3, we have:

$$[\![\mathcal{K}^{\mathrm{cn}(\Gamma_i)}_{\triangledown}(P)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_i]\!]_h$$

Then observe that

$$[\![\mathcal{R}^{N}_{\mathcal{G}}(P)]\!]_h \mid [\![\mathcal{K}^{N}_{\mathcal{G}}(P)]\!]_h \vdash_{\mathtt{H}} \Gamma_i \mid [\![\Gamma_1]\!]_h \mid \cdots [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{i+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h \equiv [\![\Gamma_1]\!]_h \mid \cdots [\![\Gamma_n]\!]_h$$

Observe that $[\![\mathcal{R}^{N}_{\mathcal{G}}(P)]\!]_h \mid [\![\mathcal{K}^{N}_{\mathcal{G}}(P)]\!]_h = [\![\mathcal{R}^{N}_{\mathcal{G}}(P) \mid \mathcal{K}^{N}_{\mathcal{G}}(P)]\!]_h$, and so we have

$$[\![\mathcal{R}^{N}_{\mathcal{G}}(P) \mid \mathcal{K}^{N}_{\mathcal{G}}(P)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$

which concludes our proof. □

Lemma B.1.2. $[\![P]\!]_h \vdash_{H} \mathcal{G} \implies [\![\mathcal{R}^{N}_{\triangledown}(P)]\!]_h \vdash_{H} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$

Where

- $\mathcal{G} = [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$
- $N = \mathrm{cn}(\Gamma_i) \cup \mathrm{cn}(\Gamma_{i+1}) \cup \cdots \cup \mathrm{cn}(\Gamma_j)$
- $\triangledown$ is any proof of $[\![P]\!]_h \vdash_{\mathtt{H}} \mathcal{G}$

Proof. By structural induction on P.

1. $P = \overline{x}\langle y\rangle.P'$.
   $[\![P]\!]_h = \overline{x}(z).([z \leftrightarrow y] \mid [\![P']\!]_h)$.
   $\triangledown$ must be:

$$\dfrac{\dfrac{\dfrac{}{[z \leftrightarrow y] \vdash_{\mathtt{H}} z : A, y : \overline{A}} \qquad \dfrac{\vdots}{[\![P']\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{k-1}]\!]_h \mid \Delta_2, x : S \mid [\![\Gamma_{k+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h \triangleq \mathcal{G}^*}}{[z \leftrightarrow y] \mid [\![P']\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{k-1}]\!]_h \mid z : A, y : \overline{A} \mid \Delta_2, x : S \mid [\![\Gamma_{k+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h}}{\overline{x}(z).([z \leftrightarrow y] \mid [\![P']\!]_h) \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h}$$

   Where $[\![\Gamma_k]\!]_h = y : \overline{A}, \Delta_2, x : A \otimes S$.
   We have two cases:

   (a) $x \in N$.
   Now $i \leq k \leq j$, as for $x \in N$, we must have $N = \mathrm{cn}(\Gamma_i) \cup \cdots \cup \mathrm{cn}(\Gamma_k) \cup \cdots \cup \mathrm{cn}(\Gamma_j)$.
   Thus we wish to show that

$$[\![\mathcal{R}^{N\setminus\{y\}}_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$

   Observe that by the inductive hypothesis:

$$[\![\mathcal{R}^{N\setminus\{y\}}_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$

   Thus we conclude this case.

(b) $x \notin N$

Now $k > j \vee k < i$, as otherwise, $i \leq k \leq j$ and then $x \in N$, as shown in the previous case. We shall assume $k > j$, but an analogous argument follows for $k < i$.

Thus we wish to show that

$$[\![\overline{x}\langle y\rangle.\mathcal{R}^{N}_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_k]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$

Oy inductive hypothesis:

$$[\![\mathcal{R}^{N}_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid \Delta_2, x : S \mid [\![\Gamma_{k+1}]\!]_h \mid [\![\Gamma_n]\!]_h$$

Then:

$$\dfrac{\dfrac{\dfrac{}{[z \leftrightarrow y] \vdash_{\mathtt{H}} z : A, y : \overline{A}} \qquad [\![\mathcal{R}^{N}_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid \Delta_2, x : S \mid [\![\Gamma_{k+1}]\!]_h \mid [\![\Gamma_n]\!]_h}{[z \leftrightarrow y] \mid [\![\mathcal{R}^{N}_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid z : A, y : \overline{A} \mid \Delta_2, x : S \mid [\![\Gamma_{k+1}]\!]_h \mid [\![\Gamma_n]\!]_h}}{\overline{x}(z).([z \leftrightarrow y] \mid [\![\mathcal{R}^{N}_{\triangledown^*}(P')]\!]_h) \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid y : \overline{A}, \Delta_2, x : A \otimes S \mid [\![\Gamma_{k+1}]\!]_h \mid [\![\Gamma_n]\!]_h}$$

Observing that $y : \overline{A}, \Delta_2, x : A \otimes S = [\![\Gamma_k]\!]_h$, and $[\![\overline{x}\langle y\rangle.\mathcal{R}^{N}_{\triangledown^*}(P')]\!]_h = \overline{x}(z).([z \leftrightarrow y] \mid [\![\mathcal{R}^{N}_{\triangledown^*}(P')]\!]_h)$ we then have:

$$[\![\overline{x}\langle y\rangle.\mathcal{R}^{N}_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_k]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$

And we conclude this case.

2. $P = x(y).P'$.

$[\![P]\!]_h = x(y).[\![P']\!]_h$.

$\triangledown$ must be:

$$\dfrac{\dfrac{\vdots}{[\![P']\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{k-1}]\!]_h \mid \Delta, y : A, x : B \mid [\![\Gamma_{k+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h \triangleq \mathcal{G}^*}}{x(y).[\![P']\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h}$$

Where $[\![\Gamma_k]\!]_h = \Delta, x : A \mathbin{⅋} B$

We have two cases:

(a) $x \in N$.

Now $i \leq k \leq j$, as for $x \in N$, we must have $N = \mathrm{cn}(\Gamma_i) \cup \cdots \cup \mathrm{cn}(\Gamma_k) \cup \cdots \cup \mathrm{cn}(\Gamma_j)$.

Thus we wish to show that:

$$\mathcal{R}^{N \cup \{y\}}_{\triangledown^*}(P') \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$

Observe that by the inductive hypothesis:

$$[\![\mathcal{R}^{N \cup \{y\}}_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$

Thus we conclude this case.

(b) $x \notin N$

Now $k > j \vee k < i$, as otherwise, $i \leq k \leq j$ and then $x \in N$, as shown in the previous case. We shall assume $k > j$ but an analogous argument follows for $k < i$.

Thus we wish to show that

$$[\![x(y).\mathcal{R}^{N}_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_k]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$

Observe by inductive hypothesis:

$$[\![\mathcal{R}^N_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid \Delta, y : A, x : B \mid [\![\Gamma_{k+1}]\!]_h \mid [\![\Gamma_n]\!]_h$$

Then:

$$\frac{[\![\mathcal{R}^N_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid \Delta, y : A, x : B \mid [\![\Gamma_{k+1}]\!]_h \mid [\![\Gamma_n]\!]_h}{x(y).[\![\mathcal{R}^N_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid \Delta, x : A \mathbin{⅋} B \mid [\![\Gamma_{k+1}]\!]_h \mid [\![\Gamma_n]\!]_h}$$

Observing that $\Delta, x : A \mathbin{⅋} B = [\![\Gamma_k]\!]_h$ and $[\![x(y).\mathcal{R}^N_{\triangledown^*}(P')]\!]_h = x(y).[\![\mathcal{R}^N_{\triangledown^*}(P')]\!]_h$, we then have:

$$[\![x(y).\mathcal{R}^N_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_k]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$

And we conclude this case.

3. Follows similarly to the previous case.
4. $P = x \triangleright \{l_i : P_i\}_{i \in I}$.

   Observe that if

   $$[\![P]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$

   We must have that $n = 1$, by the nature of applying H-$\&$.

   We have two cases:

   (a) $x \in N \vee \exists i : \exists n \in N : n \in \text{fn}(P_i)$.
   We must have $N = \text{cn}(\Gamma_1)$. This is immediate for $x \in N$. If $\exists i : \exists n \in N : n \in \text{fn}(P_i)$, observe that any name that is free in P_i must also be free in P and thus, must be present in any context that types P. Thus, it must be present in Γ_1.
   Thus, we must show that $[\![\mathcal{R}^N_{\triangledown}(P)]\!]_h = \mathbf{0} \vdash_{\mathtt{H}} \varnothing$, which is immediate.

   (b) Otherwise:
   As there is no name in Γ_1 that is in N, and there is only one context in the overall hypercontext, we have that $N = \emptyset$.
   Thus, we have to show that

   $$[\![\mathcal{R}^N_{\triangledown}(P)]\!]_h = [\![P]\!]_h \vdash_{\mathtt{H}} \Gamma_1$$

   which is immediate.

5. $P = P_1 \mid P_2$

   Consider:

   $$[\![P]\!]_h = [\![P_1]\!]_h \mid [\![P_2]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$

   $\triangledown$ must be (for some $1 \leq l \leq n$):

   $$\frac{\dfrac{\vdots}{[\![P_1]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_l]\!]_h \triangleq \mathcal{G}_1} \qquad \dfrac{\vdots}{[\![P_2]\!]_h \mid [\![\Gamma_{l+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h \triangleq \mathcal{G}_2}}{[\![P_1]\!]_h \vdash_{\mathtt{H}} [\![P_2]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h}$$

   We wish to show that:

   $$[\![\mathcal{R}^N_{\triangledown}(P)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$

   Observing

   $$[\![\mathcal{R}^N_{\triangledown}(P)]\!]_h = [\![\mathcal{R}^{N \setminus \text{fn}(P_2)}_{\triangledown_1}(P_1) \mid \mathcal{R}^{N \setminus \text{fn}(P_1)}_{\triangledown_2}(P_2)]\!]_h = [\![\mathcal{R}^{N \setminus \text{fn}(P_2)}_{\triangledown_1}(P_1)]\!]_h \mid [\![\mathcal{R}^{N \setminus \text{fn}(P_1)}_{\triangledown_2}(P_2)]\!]_h$$

   We wish to prove:

   $$[\![\mathcal{R}^{N \setminus \text{fn}(P_2)}_{\triangledown_1}(P_1)]\!]_h \mid [\![\mathcal{R}^{N \setminus \text{fn}(P_1)}_{\triangledown_2}(P_2)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$

   There are three cases:

(a) $j \leq l$:
$N \setminus \mathrm{fn}(P_2) = N$ and by inductive hypothesis we will have

$$[\![\mathcal{R}_{\triangledown_1}^{N\setminus\mathrm{fn}(P_2)}(P_1)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_k]\!]_h$$

We also have that $N \setminus \mathrm{fn}(P_1) = \emptyset$. And so by inductive hypothesis:

$$[\![\mathcal{R}_{\triangledown_2}^{N\setminus\mathrm{fn}(P_1)}(P_2)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_{k+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$

Thus,

$$\frac{\mathcal{P}_1 \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_k]\!]_h \qquad \mathcal{P}_2 \vdash_{\mathtt{H}} [\![\Gamma_{k+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h}{\mathcal{P}_1 \mid \mathcal{P}_2 \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h}$$

where for brevity we have:

$$\mathcal{P}_1 = [\![\mathcal{R}_{\triangledown_1}^{N\setminus\mathrm{fn}(P_2)}(P_1)]\!]_h$$
$$\mathcal{P}_2 = [\![\mathcal{R}_{\triangledown_2}^{N\setminus\mathrm{fn}(P_1)}(P_2)]\!]_h$$

And we conclude this case

(b) $i > l$: Analogous to case 1

(c) $i \leq l < j$:
$N \setminus \mathrm{fn}(P_2) = N_1$ where:

$$N_1 = \mathrm{cn}(\Gamma_i) \cup \cdots \cup \mathrm{cn}(\Gamma_k)$$

and by inductive hypothesis we will have

$$[\![\mathcal{R}_{\triangledown_1}^{N\setminus\mathrm{fn}(P_2)}(P_1)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h$$

We also have that $N \setminus \mathrm{fn}(P_1) = N_2$ where:

$$N_2 = \mathrm{cn}(\Gamma_{k+1}) \cup \cdots \cup \mathrm{cn}(\Gamma_j)$$

and by inductive hypothesis we will have

$$[\![\mathcal{R}_{\triangledown_2}^{N\setminus\mathrm{fn}(P_1)}(P_2)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$

Thus,

$$\frac{[\![\mathcal{R}_{\triangledown_1}^{N\setminus\mathrm{fn}(P_2)}(P_1)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \qquad [\![\mathcal{R}_{\triangledown_2}^{N\setminus\mathrm{fn}(P_1)}(P_2)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h}{[\![\mathcal{R}_{\triangledown_1}^{N\setminus\mathrm{fn}(P_2)}(P_1)]\!]_h \mid [\![\mathcal{R}_{\triangledown_2}^{N\setminus\mathrm{fn}(P_1)}(P_2)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h}$$

And we conclude this case

6. $P = (\nu xy)P'$.
Observe that $[\![P]\!]_h = (\nu xy)[\![P']\!]_h$.
$\triangledown$ must be:

$$\frac{\begin{array}{c}\vdots\\ \hline [\![P']\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{k-1}]\!]_h \mid \Delta_1, x : T \mid \Delta_2, y : \overline{T} \mid [\![\Gamma_{k+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h \triangleq \mathcal{G}^*\end{array}}{(\nu xy)[\![P']\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h}$$

where

$$\Gamma_k = \Delta_1, \Delta_2$$

Now either:

(a) $i \leq k \leq j$:
Then
$$N = \text{cn}(\Gamma_i) \cup \cdots \cup \text{cn}(\Gamma_{k-1}) \cup \text{cn}(\Delta_1) \cup \text{cn}(\Delta_2) \cup \text{cn}(\Gamma_{k+1}) \cup \cdots \cup \text{cn}(\Gamma_j)$$
Thus
$$\exists n \in N : n \in \text{cn}(\Delta_1) \cup \text{cn}(\Delta_2)$$
Thus, we must prove
$$[\![\mathcal{R}_{\triangledown^*}^{N \cup \{x,y\}}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$
which we obtain via our induction hypothesis.

(b) $k > j$:
Then
$$N = \text{cn}(\Gamma_i) \cup \cdots \text{cn}(\Gamma_j)$$
Thus
$$\neg(\exists n \in N : n \in \text{cn}(\Delta_1) \cup \text{cn}(\Delta_2))$$
Thus, we must prove
$$[\![(\nu xy)(\mathcal{R}_{\triangledown^*}^{N}(P'))]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$
From our inductive hypothesis, we obtain:
$$[\![\mathcal{R}_{\triangledown^*}^{N}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid \Delta_1, x : T \mid \Delta_2, y : \overline{T} \mid \cdots \mid [\![\Gamma_n]\!]_h$$
Subsequently, applying H-Cut:
$$(\nu xy)[\![\mathcal{R}_{\triangledown^*}^{N}(P')]\!]_h = [\![(\nu xy)[\![\mathcal{R}_{\triangledown^*}^{N}(P')]\!]_h]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid \Delta_1, \Delta_2 \mid \cdots \mid [\![\Gamma_n]\!]_h$$
Observing that $\Delta_1, \Delta_2 = \Gamma_k$ and $[\![(\nu xy)\mathcal{R}_{\triangledown^*}^{N}(P')]\!]_h = (\nu xy)[\![\mathcal{R}_{\triangledown^*}^{N}(P')]\!]_h$, we have:
$$[\![(\nu xy)\mathcal{R}_{\triangledown^*}^{N}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_k]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$
which concludes this case.

(c) $k < i$: Analogous to case 2

7. $P = \mathbf{0}$
Consider:
$$[\![\mathbf{0}]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$
Observe that this is only possible if all type assignments to Γ_i are terminating assignments.
Now, we wish to prove that
$$[\![\mathcal{R}_{\triangledown}^{N}(P)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$
Observing that $[\![\mathcal{R}_{\triangledown}^{N}(\mathbf{0})]\!]_h = [\![\mathbf{0}]\!]_h = \mathbf{0}$, we have to prove that:
$$\mathbf{0} \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{i-1}]\!]_h \mid [\![\Gamma_{j+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$
which is possible via $n - (j - i + 1)$ applications of H-$\mathbf{1}$ and $(n - (j - i + 1)) \cdot |\text{cn}(\Gamma_i)|$ applications of H-$\bot$.

$\square$

Lemma B.1.3.

$$\llbracket P \rrbracket_h \vdash_{\mathtt{H}} \llbracket \Gamma_1 \rrbracket_h \mid \cdots \mid \llbracket \Gamma_n \rrbracket_h \implies \llbracket \mathcal{K}^N(P, \mathcal{G}) \rrbracket_h \vdash_{\mathtt{H}} \llbracket \Gamma_i \rrbracket_h \mid \cdots \mid \llbracket \Gamma_j \rrbracket_h$$

Where:

- $\mathcal{G} = \llbracket \Gamma_1 \rrbracket_h \mid \cdots \mid \llbracket \Gamma_n \rrbracket_h$
- $N = \mathrm{cn}(\Gamma_i) \cup \mathrm{cn}(\Gamma_{i+1}) \cup \cdots \cup \mathrm{cn}(\Gamma_j)$

Proof. Note that $\mathcal{K}^N_\triangledown(P)$ is essentially a dual to $\mathcal{R}^N_\triangledown(P)$ and thus, the proof will be is analogous to the proof of Lemma B.1.2.

We will prove this Lemma by structural induction on P.

1. $P = \overline{x}\langle y \rangle.P'$.
   $\llbracket P \rrbracket_h = \overline{x}(z).([z \leftrightarrow y] \mid \llbracket P' \rrbracket_h)$.
   $\triangledown$ must be:

$$\dfrac{\dfrac{\overline{[z \leftrightarrow y] \vdash_{\mathtt{H}} z : A, y : \overline{A}} \qquad \dfrac{\vdots}{\llbracket P' \rrbracket_h \vdash_{\mathtt{H}} \llbracket \Gamma_1 \rrbracket_h \mid \cdots \mid \llbracket \Gamma_{k-1} \rrbracket_h \mid \Delta_2, x : S \mid \llbracket \Gamma_{k+1} \rrbracket_h \mid \cdots \mid \llbracket \Gamma_n \rrbracket_h \triangleq \mathcal{G}^*}}{[z \leftrightarrow y] \mid \llbracket P' \rrbracket_h \vdash_{\mathtt{H}} \llbracket \Gamma_1 \rrbracket_h \mid \cdots \mid \llbracket \Gamma_{k-1} \rrbracket_h \mid z : A, y : \overline{A} \mid \Delta_2, x : S \mid \llbracket \Gamma_{k+1} \rrbracket_h \mid \cdots \mid \llbracket \Gamma_n \rrbracket_h}}{\overline{x}(z).([z \leftrightarrow y] \mid \llbracket P' \rrbracket_h) \vdash_{\mathtt{H}} \llbracket \Gamma_1 \rrbracket_h \mid \cdots \mid \llbracket \Gamma_n \rrbracket_h}$$

   Where $\llbracket \Gamma_k \rrbracket_h = y : \overline{A}, \Delta_2, x : A \otimes S$.
   We have two cases:

   (a) $x \in N$.
   Now $i \leq k \leq j$, as for $x \in N$, we must have $N = \mathrm{cn}(\Gamma_i) \cup \cdots \cup \mathrm{cn}(\Gamma_k) \cup \cdots \cup \mathrm{cn}(\Gamma_j)$.
   Thus we wish to show that

$$\llbracket \overline{x}\langle y \rangle.\mathcal{K}^{N \setminus \{y\}}_{\triangledown^*}(P') \rrbracket_h \vdash_{\mathtt{H}} \llbracket \Gamma_i \rrbracket_h \mid \cdots \mid \llbracket \Gamma_k \rrbracket_h \mid \cdots \mid \llbracket \Gamma_j \rrbracket_h$$

   Observe that by the inductive hypothesis:

$$\llbracket \mathcal{K}^{N \setminus \{y\}}_{\triangledown^*}(P') \rrbracket_h \vdash_{\mathtt{H}} \llbracket \Gamma_i \rrbracket_h \mid \cdots \mid \Delta_2, x : S \mid \cdots \mid \llbracket \Gamma_j \rrbracket_h$$

   Then:

$$\dfrac{\dfrac{\overline{[z \leftrightarrow y] \vdash_{\mathtt{H}} z : A, y : \overline{A}} \qquad \llbracket \mathcal{K}^N_{\triangledown^*}(P') \rrbracket_h \vdash_{\mathtt{H}} \llbracket \Gamma_i \rrbracket_h \mid \cdots \mid \Delta_2, x : S \mid \cdots \mid \llbracket \Gamma_j \rrbracket_h}{[z \leftrightarrow y] \mid \llbracket \mathcal{R}^N_{\triangledown^*}(P') \rrbracket_h \vdash_{\mathtt{H}} \llbracket \Gamma_i \rrbracket_h \mid \cdots \mid z : A, y : \overline{A} \mid \Delta_2, x : S \mid \cdots \mid \llbracket \Gamma_j \rrbracket_h}}{\overline{x}(z).([z \leftrightarrow y] \mid \llbracket \mathcal{R}^N_{\triangledown^*}(P') \rrbracket_h) \vdash_{\mathtt{H}} \llbracket \Gamma_1 \rrbracket_h \mid \cdots \mid \llbracket \Gamma_{i-1} \rrbracket_h \mid \llbracket \Gamma_{j+1} \rrbracket_h \mid \cdots \mid y : \overline{A}, \Delta_2, x : A \otimes S \mid \llbracket \Gamma_{k+1} \rrbracket_h \mid \llbracket \Gamma_n \rrbracket_h}$$

   Observing that $y : \overline{A}, \Delta_2, x : A \otimes S = \llbracket \Gamma_k \rrbracket_h$, and $\llbracket \overline{x}\langle y \rangle.\mathcal{K}^N_{\triangledown^*}(P') \rrbracket_h = \overline{x}(z).([z \leftrightarrow y] \mid \llbracket \mathcal{K}^N_{\triangledown^*}(P') \rrbracket_h)$ we then have:

$$\overline{x}(z).([z \leftrightarrow y] \mid \llbracket \mathcal{R}^N_{\triangledown^*}(P') \rrbracket_h) \vdash_{\mathtt{H}} \llbracket \Gamma_i \rrbracket_h \mid \cdots \mid \llbracket \Gamma_k \rrbracket_h \mid \cdots \mid \llbracket \Gamma_j \rrbracket_h$$

   And we conclude this case.

(b) $x \notin N$

Now $k > j \vee k < i$, as otherwise, $i \leq k \leq j$ and then $x \in N$, as shown in the previous case.
Thus we wish to show that

$$[\![\mathcal{K}^{N}_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h$$

We shall assume $k > j$, but an analogous argument follows for $k < i$.
Observe by inductive hypothesis:

$$[\![\mathcal{K}^{N}_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h$$

Thus we conclude this case.

2. $P = x(y).P'$.
$[\![P]\!]_h = x(y).[\![P']\!]_h$.
$\triangledown$ must be:

$$\dfrac{\dfrac{\vdots}{[\![P']\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{k-1}]\!]_h \mid \Delta, y : A, x : B \mid [\![\Gamma_{k+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h \triangleq \mathcal{G}^*}}{x(y).[\![P']\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h}$$

Where $[\![\Gamma_k]\!]_h = \Delta, x : A \parr B$

We have two cases:

(a) $x \in N$.

Now $i \leq k \leq j$, as for $x \in N$, we must have $N = \mathrm{cn}(\Gamma_i) \cup \cdots \cup \mathrm{cn}(\Gamma_k) \cup \cdots \cup \mathrm{cn}(\Gamma_j)$.
Thus we wish to show that

$$[\![x(y).\mathcal{K}^{N\cup\{y\}}_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_k]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h$$

Observe by inductive hypothesis:

$$[\![\mathcal{K}^{N\cup\{y\}}_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid \Delta, y : A, x : B \mid [\![\Gamma_j]\!]_h$$

Then:

$$\dfrac{[\![\mathcal{K}^{N\cup\{y\}}_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid \Delta, y : A, x : B \mid [\![\Gamma_j]\!]_h}{[\![x(y).\mathcal{K}^{N\cup\{y\}}_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid \Delta, x : A \parr B \mid [\![\Gamma_j]\!]_h}$$

Observing that $\Delta, x : A \parr B = [\![\Gamma_k]\!]_h$, we then have:

$$[\![x(y).\mathcal{K}^{N\cup\{y\}}_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_k]\!]_h \mid [\![\Gamma_j]\!]_h$$

And we conclude this case.

(b) $x \notin N$

Now $k > j \vee k < i$, as otherwise, $i \leq k \leq j$ and then $x \in N$, as shown in the previous case. We shall assume $k > j$ but an analogous argument follows for $k < i$.
Thus we wish to show that:

$$[\![\mathcal{K}^{N}_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h$$

Observe that by the inductive hypothesis:

$$[\![\mathcal{K}^{N}_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h$$

Thus we conclude this case.

3. Follows similarly to the previous case.
4. $P = x \triangleright \{l_i : P_i\}_{i \in I}$.

   Observe that if
   $$[\![P]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$
   We must have that $n = 1$, by the nature of applying H-&.

   We have two cases:

   (a) $x \in N \vee \exists i : \exists n \in N : n \in \text{fn}(P_i)$.
   We must have $N = \text{cn}(\Gamma_1)$. This is immediate for $x \in N$. If $\exists i : \exists n \in N : n \in \text{fn}(P_i)$, observe that any name that is free in P_i must also be free in P and thus, must be present in any context that types P. Thus, it must be present in Γ_1.
   Thus, we have to show that
   $$[\![\mathcal{K}^N_\triangledown(P)]\!]_h = [\![P]\!]_h \vdash_{\mathtt{H}} \Gamma_1$$
   which is immediate.

   (b) Otherwise:
   As there is no name in Γ_1 that is in N, and there is only one context in the overall hypercontext, we have that $N = \emptyset$.
   Thus, we must show that $[\![\mathcal{K}^N_\triangledown(P)]\!]_h = \mathbf{0} \vdash_{\mathtt{H}} \varnothing$, which is immediate.

5. $P = P_1 \mid P_2$

   Consider:
   $$[\![P]\!]_h = [\![P_1]\!]_h \mid [\![P_2]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$
   $\triangledown$ must be (for some $1 \leq l \leq n$):
   $$\frac{\dfrac{\vdots}{[\![P_1]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_l]\!]_h \triangleq \mathcal{G}_1} \qquad \dfrac{\vdots}{[\![P_2]\!]_h \mid [\![\Gamma_{l+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h \triangleq \mathcal{G}_2}}{[\![P_1]\!]_h \vdash_{\mathtt{H}} [\![P_2]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h}$$
   We wish to show that:
   $$[\![\mathcal{K}^N_\triangledown(P)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h$$
   Observing
   $$[\![\mathcal{K}^N_\triangledown(P)]\!]_h = [\![\mathcal{K}^{N \setminus \text{fn}(P_2)}_{\triangledown_1}(P_1) \mid \mathcal{K}^{N \setminus \text{fn}(P_1)}_{\triangledown_2}(P_2)]\!]_h = [\![\mathcal{K}^{N \setminus \text{fn}(P_2)}_{\triangledown_1}(P_1)]\!]_h \mid [\![\mathcal{K}^{N \setminus \text{fn}(P_1)}_{\triangledown_2}(P_2)]\!]_h$$
   We wish to prove:
   $$[\![\mathcal{K}^{N \setminus \text{fn}(P_2)}_{\triangledown_1}(P_1)]\!]_h \mid [\![\mathcal{K}^{N \setminus \text{fn}(P_1)}_{\triangledown_2}(P_2)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h$$
   There are three cases:

   (a) $j \leq l$:
   $N \setminus \text{fn}(P_2) = N$ and by inductive hypothesis we will have
   $$[\![\mathcal{K}^{N \setminus \text{fn}(P_2)}_{\triangledown_1}(P_1)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h$$
   We also have that $N \setminus \text{fn}(P_1) = \emptyset$. And so by inductive hypothesis:
   $$[\![\mathcal{K}^{N \setminus \text{fn}(P_1)}_{\triangledown_2}(P_2)]\!]_h \vdash_{\mathtt{H}} \varnothing$$
   Thus,
   $$\frac{[\![\mathcal{K}^{N \setminus \text{fn}(P_2)}_{\triangledown_1}(P_1)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h \qquad [\![\mathcal{K}^{N \setminus \text{fn}(P_1)}_{\triangledown_2}(P_2)]\!]_h \vdash_{\mathtt{H}} \varnothing}{[\![\mathcal{K}^{N \setminus \text{fn}(P_2)}_{\triangledown_1}(P_1)]\!]_h \mid [\![\mathcal{K}^{N \setminus \text{fn}(P_1)}_{\triangledown_2}(P_2)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h \mid \varnothing}$$
   Observing that $[\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h \mid \varnothing = [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h$ we conclude this case

(b) $i > l$: Analogous to case 1

(c) $i \leq l < j$:
$N \setminus \text{fn}(P_2) = N_1$ where:

$$N_1 = \text{cn}(\Gamma_i) \cup \cdots \cup \text{cn}(\Gamma_l)$$

and by inductive hypothesis we will have

$$[\![\mathcal{K}_{\triangledown_1}^{N\setminus\text{fn}(P_2)}(P_1)]\!]_h \vdash_{\texttt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_l]\!]_h$$

We also have that $N \setminus \text{fn}(P_1) = N_2$ where:

$$N_2 = \text{cn}(\Gamma_{l+1}) \cup \cdots \cup \text{cn}(\Gamma_j)$$

and by inductive hypothesis we will have

$$[\![\mathcal{K}_{\triangledown_2}^{N\setminus\text{fn}(P_1)}(P_2)]\!]_h \vdash_{\texttt{H}} [\![\Gamma_{l+1}]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h$$

Thus,

$$\frac{[\![\mathcal{K}_{\triangledown_1}^{N\setminus\text{fn}(P_2)}(P_1)]\!]_h \vdash_{\texttt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_l]\!]_h \qquad [\![\mathcal{K}_{\triangledown_2}^{N\setminus\text{fn}(P_1)}(P_2)]\!]_h \vdash_{\texttt{H}} [\![\Gamma_{l+1}]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h}{[\![\mathcal{K}_{\triangledown_1}^{N\setminus\text{fn}(P_2)}(P_1)]\!]_h \mid [\![\mathcal{K}_{\triangledown_2}^{N\setminus\text{fn}(P_1)}(P_2)]\!]_h \vdash_{\texttt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_l]\!]_h \mid [\![\Gamma_{l+1}]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h}$$

Observing $[\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_l]\!]_h \mid [\![\Gamma_{l+1}]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h = [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h$ we conclude this case

6. $P = (\nu xy)P'$. Observe that $[\![P]\!]_h = (\nu xy)[\![P']\!]_h$.
$\triangledown$ must be:

$$\cfrac{\cfrac{\vdots}{[\![P']\!]_h \vdash_{\texttt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_{k-1}]\!]_h \mid \Delta_1, x : T \mid \Delta_2, y : \overline{T} \mid [\![\Gamma_{k+1}]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h \triangleq \mathcal{G}^*}}{(\nu xy)[\![P']\!]_h \vdash_{\texttt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h}$$

where

$$\Gamma_k = \Delta_1, \Delta_2$$

Now either:

(a) $i \leq k \leq j$:
Then

$$N = \text{cn}(\Gamma_i) \cup \cdots \cup \text{cn}(\Gamma_{k-1}) \cup \text{cn}(\Delta_1) \cup \text{cn}(\Delta_2) \cup \text{cn}(\Gamma_{k+1}) \cup \cdots \cup \text{cn}(\Gamma_j)$$

Thus

$$\exists n \in N : n \in \text{cn}(\Delta_1) \cup \text{cn}(\Delta_2)$$

Thus, we must prove

$$[\![(\nu xy)\mathcal{K}_{\triangledown^*}^{N\cup\{x,y\}}(P')]\!]_h \vdash_{\texttt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_k]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h$$

From our inductive hypothesis, we obtain:

$$[\![\mathcal{K}_{\triangledown^*}^{N\cup\{x,y\}}(P')]\!]_h \vdash_{\texttt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid \Delta_1, x : T \mid \Delta_2, y : \overline{T} \mid \cdots \mid [\![\Gamma_j]\!]_h$$

Subsequently, applying H-Cut:

$$(\nu xy)[\![\mathcal{K}_{\triangledown^*}^{N\cup\{x,y\}}(P')]\!]_h \vdash_{\texttt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid \Delta_1, \Delta_2 \mid \cdots \mid [\![\Gamma_j]\!]_h$$

Observing that $\Delta_1, \Delta_2 = \Gamma_k$ and $[\![(\nu xy)\mathcal{K}_{\triangledown^*}^{N\cup\{x,y\}}(P')]\!]_h = (\nu xy)[\![\mathcal{K}_{\triangledown^*}^{N\cup\{x,y\}}(P')]\!]_h$, we have:

$$[\![(\nu xy)\mathcal{K}_{\triangledown^*}^{N\cup\{x,y\}}(P')]\!]_h \vdash_{\texttt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_k]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h$$

which concludes this case.

(b) $k > j$:
Then

$$N = \text{cn}(\Gamma_i) \cup \cdots \text{cn}(\Gamma_j)$$

Thus

$$\neg(\exists n \in N : n \in \text{cn}(\Delta_1) \cup \text{cn}(\Delta_2))$$

Thus, we must prove

$$[\![\mathcal{K}^N_{\triangledown^*}(P')]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h$$

which we obtain via our induction hypothesis.

(c) $k < i$: Analogous to case 2

7. $P = \mathbf{0}$

Consider:

$$[\![\mathbf{0}]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_1]\!]_h \mid \cdots \mid [\![\Gamma_n]\!]_h$$

Observe that this is only possible if all type assignments to Γ_i are terminating assignments.

Thus, we wish to prove that

$$[\![\mathcal{K}^N_{\triangledown}(P)]\!]_h \vdash_{\mathtt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h$$

Observing that $[\![\mathcal{K}^N_{\triangledown}(P)]\!]_h = [\![\mathbf{0}]\!]_h = \mathbf{0}$, we have to prove that:

$$\mathbf{0} \vdash_{\mathtt{H}} [\![\Gamma_i]\!]_h \mid \cdots \mid [\![\Gamma_j]\!]_h$$

which is possible via $j - i + 1$ applications of H-**1** and $(j - i + 1) \cdot |\text{cn}(\Gamma_i)|$ applications of H-$\perp$

□

Lemma B.1.4.

$$([\![P]\!]_h \vdash_{H} \mathcal{G} \wedge [\![P']\!]_h \vdash_{H} \mathcal{G} \wedge [\![D[P]]\!]_h \vdash_{H} \mathcal{G}') \implies [\![D[P']]\!]_h \vdash_{H} \mathcal{G}'$$

Proof (sketch).

Induction on D. As the inductive hypothesis preserves the context for $D'[P']$, the result context of applying the action on $D'[P']$ will be the same as applying the action on $D'[P]$.