# Taking Linear Logic Apart

Wen Kokke
University of Edinburgh
Edinburgh, Scotland
wen.kokke@ed.ac.uk

Fabrizio Montesi
University of Southern Denmark
Odense, Denmark
fmontesi@imada.sdu.dk

Marco Peressotti
University of Southern Denmark
Odense, Denmark
peressotti@imada.sdu.dk

Process calculi based on logic, such as πDILL and CP, provide a foundation for deadlock-free concurrent programming. However, in previous work, there is a mismatch between the rules for constructing proofs and the term constructors of the π-calculus: the fundamental operator for parallel composition does not correspond to any rule of linear logic.

Kokke *et al*. [12] introduced Hypersequent Classical Processes (HCP), which addresses this mismatch using hypersequents (collections of sequents) to register parallelism in the typing judgements. However, the step from CP to HCP is a big one. As of yet, HCP does not have reduction semantics, and the addition of delayed actions means that CP processes interpreted as HCP processes do not behave as they would in CP.

We introduce HCP$^-$, a variant of HCP with reduction semantics and without delayed actions. We prove progress, preservation, and termination, and show that HCP$^-$ supports the same communication protocols as CP.

## 1 Introduction

Classical Processes (CP) [20] is a process calculus inspired by the correspondence between the session-typed π-calculus and linear logic [5], where processes correspond to proofs, session types (communication protocols) to propositions, and communication to cut elimination. This correspondence allows for exchanging methods between the two fields. For example, the proof theory of linear logic can be used to guarantee progress for processes [5, 20].

The main attraction of CP is that its semantics are *prescribed* by the cut elimination procedure of Classical Linear Logic (CLL). This permits us to reuse the metatheory of linear logic "as is" to reason about the behaviour of processes. However, there is a mismatch between the structure of the proof terms of CLL and the term constructs of the standard π-calculus [16, 17]. For instance, the term for output of a linear name is $x[y].(P \mid Q)$, which is read "send $y$ over $x$ and proceed as $P$ in parallel to $Q$". Note that this is a single term constructor, which takes all four arguments at the same time. This is caused by directly adopting the $(\otimes)$-rule from CLL as the process calculus construct for sending: the $(\otimes)$-rule has two premises (corresponding to $P$ and $Q$ in the output term), and checks that they share no resources (in the output term, $y$ can be used only by $P$, and $x$ can be used only by $Q$).

There is no independent parallel term $(P \mid Q)$ in the grammar of CP terms. Instead, parallel composition shows up in any term which corresponds to a typing rule which splits the context. Even if we were to add an independent parallel composition via the MIX-rule, as suggested in the original presentation of CP [20], there would be no way to allow the composed process $P$ and $Q$ to communicate as in the standard π-calculus, as there is no independent name restriction either! Instead, synchronisation is governed by the "cut" operator $(\nu x)(P \mid Q)$, which composes $P$ and $Q$, enabling them to communicate along $x$. Worse, if we naively add an independent parallel composition as well as a name restriction, using the

This is a revision of a paper with the same title,
published in EPTCS 292, 2019, pp.90–103,
doi:10.4204/EPTCS.292.5

rules shown below, we lose cut elimination, and therefore deadlock-freedom!

$$\frac{P \vdash \Gamma \qquad Q \vdash \Delta}{P \mid Q \vdash \Gamma, \Delta}\ \textsc{Mix} \qquad\qquad \frac{P \vdash \Gamma, x : A, y : A^{\perp}}{(\nu xy)P \vdash \Gamma}\ \text{“}\textsc{Cut}\text{”}$$

This syntactic mismatch has an effect on the semantics as well. For instance, the $\beta$-reduction for output and input in CP is $(\nu x)(x[y].(P \mid Q) \mid x(y).R) \Longrightarrow (\nu y)(P \mid (\nu x)(Q \mid R))$. Here, the parallel composition $(P \mid Q)$ is of no relevance to this communication, yet the rule needs to inspect it to be able to nest the name restrictions appropriately in the resulting term.

Kokke *et al.* [12] introduced Hypersequent Classical Processes (HCP), which addresses this mismatch. The key insight is to register parallelism in the typing judgements using hypersequents [2], a technique from logic which generalises judgements from one sequent to many. This allows us to take apart the term constructs used in Classical Processes (CP) to more closely match those of the standard π-calculus. HCP has labelled transition semantics with delayed actions [15] and a full abstraction result: bisimilarity, denotational equivalence, and barbed congruence coincide.

However, the step from CP to HCP is a big one. As of yet, HCP does not have reduction semantics, and the addition of delayed actions means that CP processes interpreted as HCP processes do not behave as they would in CP.

In this paper, we address these issues by introducing HCP$^-$, a variant of HCP with reduction semantics and without delayed actions. We proceed as follows. We start by introducing CP (Section 2). Then, we introduce our variant of HCP$^-$ and prove it enjoys subject reduction and progress (Section 3). We prove that every CP process is an HCP$^-$ process, relate processes in HCP$^-$ back to CP, and prove that HCP$^-$ supports the same communication protocols as CP (Section 4). Finally, we discuss related work (Section 5).

## 2 Classical Processes

In this section, we introduce CP. In order to keep the discussion of HCP$^-$ in Section 3 simple, we restrict ourselves to the multiplicative-additive subset of CP. We foresee no problems in extending the proofs in Section 3 to cover the remaining features of CP (polymophism and the exponentials).

### 2.1 Terms

The term language of CP is a variant of the π-calculus. The variables $x$, $y$, and $z$ range over channel names. The construct $x {\leftrightarrow} y$ links two channels [4, 19], forwarding messages received on $x$ to $y$ and vice versa. The construct $(\nu x)(P \mid Q)$ creates a new channel $x$, and composes two processes, $P$ and $Q$, which communicate on $x$, in parallel. Therefore, in $(\nu x)(P \mid Q)$ the name $x$ is bound in both $P$ and $Q$. In $x(y).P$ and $x[y].(P \mid Q)$, round brackets denote input, square brackets denote output. CP uses bound output [19], meaning that both input and output bind a new name. In $x(y).P$ the new name $y$ is bound in $P$. In $x[y].(P \mid Q)$, the new name $y$ is only bound in $P$, while $x$ is only bound in $Q$.

**Definition 2.1** (Terms)**.** *Process terms are given by the following grammar:*

$$\begin{array}{rllcll}
P,Q,R ::= & x{\leftrightarrow}y & \textit{link} & \mid & (\nu x)(P \mid Q) & \textit{parallel composition, “cut”} \\
\mid & x[y].(P \mid Q) & \textit{output} & \mid & x(y).P & \textit{input} \\
\mid & x[].0 & \textit{halt} & \mid & x().P & \textit{wait} \\
\mid & x \triangleleft \texttt{inl}.P & \textit{select left choice} & \mid & x \triangleleft \texttt{inr}.P & \textit{select right choice} \\
\mid & x \triangleright \{\texttt{inl} : P; \texttt{inr} : Q\} & \textit{offer binary choice} & \mid & x \triangleright \{\} & \textit{offer nullary choice}
\end{array}$$

Terms in CP are identified up to structural congruence, which states that links are symmetric, and parallel compositions $(\nu x)(P \mid Q)$ are associative and commutative.

**Definition 2.2** (Structural congruence)**.** *The structural congruence $\equiv$ is the congruence closure over terms which satisfies the following additional axioms:*

$$
\begin{array}{lll}
(\leftrightarrow\text{-}sym) & x{\leftrightarrow}y & \equiv\ y{\leftrightarrow}x \\
(\nu\text{-}comm) & (\nu x)(P \mid Q) & \equiv\ (\nu x)(Q \mid P) \\
(\nu\text{-}assoc) & (\nu x)(P \mid (\nu y)(Q \mid R)) & \equiv\ (\nu y)((\nu x)(P \mid Q) \mid R) \ \ \text{if } x \notin R \text{ and } y \notin P
\end{array}
$$

The reduction semantics presented here are a variant of those presented by Lindley and Morris [13], who showed that reduction in CP can be decomposed in two phases: one in which all β-reduction happens, and one in which an arbitrary action, blocked on an external communication, is moved to the top of the term using commuting conversions. We choose to stop after the first phase, and do away with the commuting conversions.

Reductions relate processes with their reduced forms *e.g.*, a reduction $P \Longrightarrow Q$ denotes that the process $P$ can reduce to the process $Q$ in a single step.

**Definition 2.3** (Reduction)**.** *Reductions are described by the smallest relation $\Longrightarrow$ on process terms closed under rules below.*

$$
\begin{array}{llll}
(\leftrightarrow) & (\nu x)(w{\leftrightarrow}x \mid P) & \Longrightarrow & P\{w/x\} \\
(\beta\otimes\parr) & (\nu x)(x[y].(P \mid Q) \mid x(y).R) & \Longrightarrow & (\nu y)(P \mid (\nu x)(Q \mid R)) \\
(\beta\mathbf{1}\bot) & (\nu x)(x[].0 \mid x().P) & \Longrightarrow & P \\
(\beta\oplus\&_1) & (\nu x)(x \triangleleft \mathtt{inl}.P \mid x \triangleright \{\mathtt{inl} : Q; \mathtt{inr} : R\}) & \Longrightarrow & (\nu x)(P \mid Q) \\
(\beta\oplus\&_2) & (\nu x)(x \triangleleft \mathtt{inr}.P \mid x \triangleright \{\mathtt{inl} : Q; \mathtt{inr} : R\}) & \Longrightarrow & (\nu x)(P \mid R)
\end{array}
$$

$$
\frac{P \Longrightarrow P'}{(\nu x)(P \mid Q) \Longrightarrow (\nu x)(P' \mid Q)}\ (\gamma\nu)
\qquad
\frac{P \equiv Q \qquad Q \Longrightarrow Q' \qquad Q' \equiv P'}{P \Longrightarrow P'}\ (\gamma\equiv)
$$

*Relations $\Longrightarrow^+$ and $\Longrightarrow^\star$ are the transitive, and the reflexive, transitive closures of $\Longrightarrow$, respectively.*

Note that we do not need to add a side condition to $(\beta\otimes\parr)$ to restrict its usage to the case where $y$ is bound in $P$ and $x$ is bound in $Q$, as this is required by the definition of the send construct $x[y].(P \mid Q)$.

## 2.2 Types

Channels in CP are typed using a session type system which corresponds to classical linear logic.

**Definition 2.4** (Types)**.**

$$
\begin{array}{rlll|lll}
A,B,C ::= & A \otimes B & \textit{pair of independent processes} & & \mathbf{1} & \textit{unit for } \otimes \\
\mid & A \parr B & \textit{pair of interdependent processes} & \mid & \bot & \textit{unit for } \parr \\
\mid & A \oplus B & \textit{internal choice} & \mid & \mathbf{0} & \textit{unit for } \oplus \\
\mid & A \,\&\, B & \textit{external choice} & \mid & \top & \textit{unit for } \&
\end{array}
$$

A channel of type $A \otimes B$ represents a pair of channels, which communicate with two independent processes—that is to say, two processes who share no channels. A process acting on a channel of type $A \otimes B$ will send one endpoint of a fresh channel, and then split into a pair of independent processes. One of these processes will be responsible for an interaction of type $A$ over the fresh channel, while the other process continues to interact as $B$.

A channel of type $A \parr B$ represents a pair of interdependent channels, which are used within a single process. A process acting on a channel of type $A \parr B$ will receive a channel to act on, and communicate on its channels in whatever order it pleases. This means that the usage of one channel can depend on that of another—*e.g.*, the interaction of type $B$ could depend on the result of the interaction of type $A$, or vise versa, and if $A$ and $B$ are complex types, their interactions could likewise interweave in complex ways.

A process acting on a channel of type $A \oplus B$ either sends the value inl to select an interaction of type $A$ or the value inr to select one of type $B$. A process acting on a channel of type $A \,\&\, B$ receives such a value, and then offers an interaction of either type $A$ or $B$, correspondingly.

Duality plays a crucial role in both linear logic and session types. In CP, the two endpoints of a channel are assigned dual types. This ensures that, for instance, whenever a process *sends* across a channel, the process on the other end of that channel is waiting to *receive*. Each type $A$ has a dual, written $A^\perp$. Duality is an involutive function *i.e.*, $(A^\perp)^\perp = A$.

**Definition 2.5** (Duality)**.**

$$\begin{array}{lcl lcl lcl lcl}
(A \otimes B)^\perp & = & A^\perp \parr B^\perp & \mathbf{1}^\perp & = & \perp & (A \parr B)^\perp & = & A^\perp \otimes B^\perp & \perp^\perp & = & \mathbf{1} \\
(A \oplus B)^\perp & = & A^\perp \,\&\, B^\perp & \mathbf{0}^\perp & = & \top & (A \,\&\, B)^\perp & = & A^\perp \oplus B^\perp & \top^\perp & = & \mathbf{0}
\end{array}$$

An environment associates channels with types. Names in environments must be unique, and two environments $\Gamma$ and $\Delta$ can only be combined as $\Gamma,\Delta$ if $\mathrm{fv}(\Gamma) \cap \mathrm{fv}(\Delta) = \varnothing$.

**Definition 2.6** (Environments)**.** $\Gamma,\Delta,\Theta ::= \cdot \mid \Gamma, x : A$

A typing judgement associates a process with collections of typed channels.

**Definition 2.7** (Typing judgements)**.** *A typing judgement* $P \vdash x_1 : A_1, \ldots, x_n : A_n$ *denotes that the process* $P$ *communicates along channels* $x_1$, …, $x_n$ *following protocols* $A_1$, …, $A_n$. *Typing judgements are derived using rules below.*

*Structural rules*

$$\frac{}{x{\leftrightarrow}y \vdash x : A, y : A^\perp}\ \textsc{Ax} \qquad \frac{P \vdash \Gamma, x : A \qquad Q \vdash \Delta, x : A^\perp}{(\nu x)(P \mid Q) \vdash \Gamma, \Delta}\ \textsc{Cut}$$

*Logical rules*

$$\frac{P \vdash \Gamma, y : A \qquad Q \vdash \Delta, x : B}{x[y].(P \mid Q) \vdash \Gamma, \Delta, x : A \otimes B}\ (\otimes) \qquad \frac{P \vdash \Gamma, y : A, x : B}{x(y).P \vdash \Gamma, x : A \parr B}\ (\parr)$$

$$\frac{P \vdash \Gamma}{x().P \vdash \Gamma, x : \perp}\ (\perp) \qquad \frac{}{x[].0 \vdash x : \mathbf{1}}\ (\mathbf{1})$$

$$\frac{P \vdash \Gamma, x : A}{x \triangleleft \mathtt{inl}.P \vdash \Gamma, x : A \oplus B}\ (\oplus_1) \qquad \frac{P \vdash \Gamma, x : B}{x \triangleleft \mathtt{inr}.P \vdash \Gamma, x : A \oplus B}\ (\oplus_2)$$

$$\frac{P \vdash \Gamma, x : A \qquad Q \vdash \Gamma, x : B}{x \triangleright \{\mathtt{inl} : P; \mathtt{inr} : Q\} \vdash \Gamma, x : A \,\&\, B}\ (\&)$$

$$\textit{(no rule for } \mathbf{0}\textit{)} \qquad \frac{}{x \triangleright \{\} \vdash \Gamma, x : \top}\ (\top)$$

### 2.3 Metatheory

CP enjoys subject reduction, termination, and progress [13, 20].

**Lemma 2.8** (Preservation for $\equiv$). *If $P \equiv Q$, then $P \vdash \Gamma$ iff $Q \vdash \Gamma$.*

*Proof.* By induction on the derivation of $P \equiv Q$. □

**Theorem 2.9** (Preservation). *If $P \vdash \Gamma$ and $P \Longrightarrow Q$, then $Q \vdash \Gamma$.*

*Proof.* By induction on the derivation of $P \Longrightarrow Q$. □

**Definition 2.10** (Actions). *A process $P$ acts on $x$ whenever $x$ is free in the outermost term constructor of $P$,* e.g.*, $x[y].(P \mid Q)$ acts on $x$ but not on $y$, and $x \leftrightarrow y$ acts on both $x$ and $y$. A process $P$ is an action if it acts on some channel $x$.*

**Definition 2.11** (Canonical forms). *A process $P$ is in canonical form if*

$$P \equiv (\nu x_1)(P_1 \mid \ldots (\nu x_n)(P_n \mid P_{n+1}) \ldots),$$

*such that: no process $P_i$ is a cut; no process $P_i$ is a link acting on a bound channel $x_i$; and no two processes $P_i$ and $P_j$ are acting on the same bound channel $x_i$.*

**Corollary 2.12.** *If a process $P$ is in canonical form, then it is blocked on an external communication.*

*Proof.* We have $P \equiv (\nu x_1)(P_1 \mid \ldots (\nu x_n)(P_n \mid P_{n+1}) \ldots)$ such that no $P_i$ is a cut or a link, and no two processes $P_i$ and $P_j$ are acting on the same bound channel. The prefix of cuts introduces $n$ channels, and $n+1$ processes. Therefore, at least *one* of the processes $P_i$ must be acting on a free channel, i.e., blocked on an external communication. □

**Theorem 2.13** (Progress). *If $P \vdash \Gamma$, then either $P$ is in canonical form, or there exists a process $Q$ such that $P \Longrightarrow Q$.*

*Proof.* We consider the maximum prefix of cuts of $P$ such that $P \equiv (\nu x_1)(P_1 \mid \ldots (\nu x_n)(P_n \mid P_{n+1}) \ldots)$ and no $P_i$ is a cut. If any process $P_i$ is a link, we reduce by $(\leftrightarrow)$. If any two processes $P_i$ and $P_j$ are acting on the same channel $x_i$, we rewrite by $\equiv$ and reduce by the appropriate $\beta$-rule. Otherwise, $P$ is in canonical form. □

**Theorem 2.14** (Termination). *If $P \vdash \Gamma$, then there are no infinite $\Longrightarrow$-reduction sequences.*

*Proof.* Every reduction reduces a single cut to zero, one or two cuts. However, each of these cuts is smaller, measured in the size of the cut formula. Furthermore, each instance of the structural congruence preserves the size of the cut. Therefore, there cannot be an infinite $\Longrightarrow$-reduction sequence. □

## 3 Hypersequent Classical Processes

We introduce our variant of Hypersequent Classical Processes (HCP$^-$), itself a variant of CP which registers parallelism in the typing judgements using hypersequents, allowing us to take apart the monolithic term constructors of CP (*e.g.*, $x[y].(P \mid Q)$) into the corresponding π-calculus term constructs.

The crucial difference between HCP$^-$ as described here and HCP as described by Kokke *et al.* [12] is in the absence of delayed actions. However, removing delayed actions from HCP introduces self-locking processes, which we rule out by adopting (with minor changes) the type system first introduced by Montesi and Peressotti in [18] (see Section 3.2). Differently from *loc. cit.*, we follow CP in using the same name for both endpoints of a channel, writing, *e.g.*, $(\nu x)(x[].0 \mid x().P)$ as opposed to $(\nu xy)(x[].0 \mid y().P)$.

### 3.1 Terms

The term language of HCP$^-$ is a variant of CP where the term constructs have been taken apart into primitives which more closely resemble the π-calculus primitives.

**Definition 3.1** (Terms)**.**

$$
\begin{array}{rlll|lll}
P,Q,R ::= & x{\leftrightarrow}y & \textit{link} & & 0 & \textit{terminated process} \\
\mid & (\nu x)P & \textit{name restriction, “cut”} & \mid & (P \mid Q) & \textit{parallel composition, “mix”} \\
\mid & x[y].P & \textit{output} & \mid & x(y).P & \textit{input} \\
\mid & x[].P & \textit{halt} & \mid & x().P & \textit{wait} \\
\mid & x \triangleleft \texttt{inl}.P & \textit{select left choice} & \mid & x \triangleleft \texttt{inr}.P & \textit{select right choice} \\
\mid & x \triangleright \{\texttt{inl}: P; \texttt{inr}: Q\} & \textit{offer binary choice} & \mid & x \triangleright \{\} & \textit{offer nullary choice}
\end{array}
$$

A pleasant effect of our updated syntax is that it makes our structural congruence much more standard: it has associativity, commutativity, and a unit for parallel composition, commutativity of name restrictions, and scope extrusion.

**Definition 3.2** (Structural congruence)**.** *The structural congruence $\equiv$ is the congruence closure over terms which satisfies the following additional axioms:*

$$
\begin{array}{llcllllcl}
(\leftrightarrow\text{-}\mathit{sym}) & x{\leftrightarrow}y & \equiv & y{\leftrightarrow}x & (\mathit{halt}) & P \mid 0 & \equiv & P & \\
(\mid\text{-}\mathit{comm}) & P \mid Q & \equiv & Q \mid P & (\mid\text{-}\mathit{assoc}) & P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R & \\
(\nu\text{-}\mathit{comm}) & (\nu x)(\nu y)P & \equiv & (\nu y)(\nu x)P & (\mathit{scope\text{-}ext}) & (\nu x)(P \mid Q) & \equiv & P \mid (\nu x)Q & \text{if } x \notin P
\end{array}
$$

There are two changes to the reduction system. First, since $x[y].P$ and $x[].P$ are now terms in their own right, the $(\beta\otimes\invamp)$ and $(\beta\mathbf{1}\bot)$ rules are simpler. Second, since we decomposed $(\nu x)(P \mid Q)$ into an independent name restriction and parallel composition, the relevant $\gamma$-rule all decompose as well.

**Definition 3.3** (Reduction)**.** *Reductions are described by the smallest relation $\Longrightarrow$ on process terms closed under the rules below:*

$$
\begin{array}{lllll}
(\leftrightarrow) & (\nu x)(w{\leftrightarrow}x \mid P) & \Longrightarrow & P\{w/x\} \\
(\beta\otimes\invamp) & (\nu x)(x[y].P \mid x(y).R) & \Longrightarrow & (\nu x)(\nu y)(P \mid R) \\
(\beta\mathbf{1}\bot) & (\nu x)(x[].P \mid x().Q) & \Longrightarrow & P \mid Q \\
(\beta\oplus\&_1) & (\nu x)(x \triangleleft \texttt{inl}.P \mid x \triangleright \{\texttt{inl}: Q; \texttt{inr}: R\}) & \Longrightarrow & (\nu x)(P \mid Q) \\
(\beta\oplus\&_2) & (\nu x)(x \triangleleft \texttt{inr}.P \mid x \triangleright \{\texttt{inl}: Q; \texttt{inr}: R\}) & \Longrightarrow & (\nu x)(P \mid R)
\end{array}
$$

$$
\frac{P \Longrightarrow P'}{(\nu x)P \Longrightarrow (\nu x)P'}\,(\gamma\nu)
\qquad
\frac{P \Longrightarrow P'}{P \mid Q \Longrightarrow P' \mid Q}\,(\gamma\mid)
\qquad
\frac{P \equiv Q \qquad Q \Longrightarrow Q' \qquad Q' \equiv P'}{P \Longrightarrow P'}\,(\gamma\equiv)
$$

*Relations $\Longrightarrow^+$ and $\Longrightarrow^\star$ are the transitive, and the reflexive, transitive closures of $\Longrightarrow$, respectively.*

### 3.2 Types

We use the same definitions for types and environments for HCP$^-$ as we used for CP. However, we introduce a new layer on top of sequents: hypersequents. As CP is a one-sided logic, and it uses the left-hand side of the turnstile to write the process, the traditional hypersequent notation can look confusing: “$P \vdash \Gamma_1 \mid \ldots \mid \ \vdash \Gamma_n$” seems to claim that $P$ acts according to protocol $\Gamma_1$. What are all the other Γs doing there? Are they typing empty processes? Therefore, we opt to leave out the repeated turnstile, and instead work with the notion of "hyper-environments". However, we will still refer to our system as a hypersequent system. A hyper-environment is either empty, or consist of a series of typing environments, separated by vertical bars. A hyper-environment $\Gamma_1 \mid \ldots \mid \Gamma_n$ types a series of $n$ entangled, but independent processes.

**Definition 3.4** (Hyper-environments). $\mathcal{G}, \mathcal{H} ::= \varnothing \mid \mathcal{G} \mid \Gamma$

A hyper-environment is a multiset of environments. While names within environments must be unique, names may be shared between multiple environments in a hyper-environment. We write $\mathcal{G} \mid \mathcal{H}$ to combine two hyper-environments.

Typing judgements in HCP$^{-}$ associate processes with hyper-environments. H-MIX composes two processes in parallel, but remembers that they are independent in the sequent. H-CUT and ($\otimes$) take as their premise a process which consists of at least two independent processes, and connects them, eliminating the vertical bar (the side condition $x \notin \mathcal{G}$ ensures that sessions are binary).

**Definition 3.5** (Typing judgements). *A typing judgement $P \vdash \Gamma_1 \mid \ldots \mid \Gamma_n$ denotes that the process $P$ consists of $n$ independent, but potentially entangled processes, each of which communicates according to its own protocol $\Gamma_i$. Typing judgements can be constructed using the inference rules below.*

*Structural rules*

$$\dfrac{}{x \leftrightarrow y \vdash x : A, y : A^{\perp}}\ \textsc{Ax} \qquad \dfrac{P \vdash \mathcal{G} \mid \Gamma, x : A \mid \Delta, x : A^{\perp} \qquad x \notin \mathcal{G}}{(\nu x)P \vdash \mathcal{G} \mid \Gamma, \Delta}\ \textsc{H-Cut}$$

$$\dfrac{P \vdash \mathcal{G} \qquad Q \vdash \mathcal{H}}{P \mid Q \vdash \mathcal{G} \mid \mathcal{H}}\ \textsc{H-Mix} \qquad \dfrac{}{0 \vdash \varnothing}\ \textsc{H-Mix}_0$$

*Logical rules*

$$\dfrac{P \vdash \Gamma, y : A \mid \Delta, x : B}{x[y].P \vdash \Gamma, \Delta, x : A \otimes B}\ (\otimes) \qquad \dfrac{P \vdash \Gamma, y : A, x : B}{x(y).P \vdash \Gamma, x : A \parr B}\ (\parr)$$

$$\dfrac{P \vdash \varnothing}{x[].P \vdash x : \mathbf{1}}\ (\mathbf{1}) \qquad \dfrac{P \vdash \Gamma}{x().P \vdash \Gamma, x : \perp}\ (\perp)$$

$$\dfrac{P \vdash \Gamma, x : A}{x \triangleleft \mathtt{inl}.P \vdash \Gamma, x : A \oplus B}\ (\oplus_1) \qquad \dfrac{P \vdash \Gamma, x : B}{x \triangleleft \mathtt{inr}.P \vdash \Gamma, x : A \oplus B}\ (\oplus_2)$$

$$\dfrac{P \vdash \Gamma, x : A \qquad Q \vdash \Gamma, x : B}{x \triangleright \{\mathtt{inl} : P; \mathtt{inr} : Q\} \vdash \Gamma, x : A \,\&\, B}\ (\&)$$

$$\textit{(no rule for } \mathbf{0}\textit{)} \qquad \dfrac{}{x \triangleright \{\} \vdash \Gamma, x : \top}\ (\top)$$

Note that the logical rules disallow hyper-environments. If they did not, the following processes would be well-typed, even though they are stuck (unless we allow delayed actions as in HCP):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{0 \vdash \varnothing}\ \textsc{H-Mix}_0}{z[].0 \vdash z : \mathbf{1}}\ (\mathbf{1})}{x[].z[].0 \vdash x : \mathbf{1} \mid z : \mathbf{1}}\ (\mathbf{1})}{x().x[].z[].0 \vdash x : \mathbf{1} \mid x : \perp, z : \mathbf{1}}\ (\perp)}{(\nu x)(x().x[].z[].0) \vdash z : \mathbf{1}}\ \textsc{H-Cut}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{0 \vdash \varnothing}\ \textsc{H-Mix}_0}{y[].0 \vdash y : \mathbf{1}}\ (\mathbf{1})}{x[].y[].0 \vdash x : \mathbf{1} \mid y : \mathbf{1}}\ (\mathbf{1}) \qquad \dfrac{\dfrac{}{x \leftrightarrow z \vdash x : \perp, z : \mathbf{1}}\ \textsc{Ax}}{y().x \leftrightarrow z \vdash x : \perp, y : \perp, z : \mathbf{1}}\ (\perp)}{(x[].y[].0 \mid y().x \leftrightarrow z) \vdash x : \mathbf{1} \mid y : \mathbf{1} \mid x : \perp, y : \perp, z : \mathbf{1}}\ \textsc{H-Mix}}{(\nu y)(x[].y[].0 \mid y().x \leftrightarrow z) \vdash x : \mathbf{1} \mid x : \perp, z : \mathbf{1}}\ \textsc{H-Cut}}{(\nu x)(\nu y)(x[].y[].0 \mid y().x \leftrightarrow z) \vdash z : \mathbf{1}}\ \textsc{H-Cut}$$

The first process is self-locking, in that it holds both endpoints of a channel in the same process, and hence communication can never happen in a semantic without delayed and self-synchronising actions. Self-locking processes can be ruled out by requiring that the two endpoints of a channel are held in different processes *i.e.*, appear in distinct hypersequents.

The second term is not self-locked, nor a deadlock where two processes are waiting on each other. However, it is definitely stuck. The problem lies with readiness: a process typed under a hyperenvironment $\Gamma_1 \mid \ldots \mid \Gamma_n$, it promises to act as, essentially, $n$ independent processes, each of which ready to act on a channel from $\Gamma_i$. However, the process $x[].y[].0$, a single process, can be typed under the hyperenvironment $x : \mathbf{1} \mid y : \mathbf{1}$, which consists of *two* environments. We can abuse this failure of readiness by asking the process to communicate on $y$, which it is incapable of doing immediately. This is not a property of the $(\mathbf{1})$ rule allowing hyper-environments. Similar examples can be constructed using the $(\otimes)$ rule.

## 3.3 Metatheory

HCP$^-$ enjoys subject reduction, termination, and progress.

**Lemma 3.6** (Preservation for $\equiv$)**.** *If $P \equiv Q$, then $P \vdash \mathscr{G}$ iff $Q \vdash \mathscr{G}$.*

*Proof.* By induction on the derivation of $P \equiv Q$. □

**Theorem 3.7** (Preservation)**.** *If $P \vdash \mathscr{G}$ and $P \Longrightarrow Q$, then $Q \vdash \mathscr{G}$.*

*Proof.* By induction on the derivation of $P \Longrightarrow Q$. □

**Definition 3.8** (Actions)**.** *A process $P$ acts on $x$ whenever $x$ is free in the outermost term constructor of $P$,* e.g.*, $x[y].(P \mid Q)$ acts on $x$ but not on $y$, and $x{\leftrightarrow}y$ acts on both $x$ and $y$. A process $P$ is an action if it acts on some channel $x$.*

**Definition 3.9** (Canonical forms)**.** *A process $P$ is in canonical form if $P \equiv (\nu x_1)\ldots(\nu x_n)(P_1 \mid \cdots \mid P_{n+m+1})$, such that: no process $P_i$ is a cut or a mix; no process $P_i$ is a link acting on a bound channel $x_i$; and no two processes $P_i$ and $P_j$ are acting on the same bound channel $x_i$.*

Note that we have added the restriction "acting on a bound channel" to the case for links. This was not necessary for CP, as all links in CP act on at least one bound channel. Consequently, processes such as $x{\leftrightarrow}y$ and $(x{\leftrightarrow}y \mid z{\leftrightarrow}w)$ are considered to be in canonical form. This is a generalisation of CP, where $x{\leftrightarrow}y$ is considered to be in canonical form. If this is objectionable, the reduction system can be extended with identity expansion, expanding, *e.g.*, the process $x^{\perp}{\leftrightarrow}y$ to $x().y[].0$.

**Corollary 3.10.** *If a process $P$ is in canonical form, then it is blocked on an external communication.*

*Proof.* We have $P \equiv (\nu x_1)\ldots(\nu x_n)(P_1 \mid \ldots \mid P_{n+m+1})$, such that no $P_i$ is a cut or a link acting on a bound channel, and no two processes $P_i$ and $P_j$ are acting on the same bound channel. The prefix of cuts and mixes introduces $n$ channels. Each application of cut requires an application of mix, so the prefix introduces $n+m+1$ processes. Therefore, at least $m+1$ of the processes $P_i$ must be acting on a free channel, i.e., blocked on an external communication. □

**Theorem 3.11** (Progress)**.** *If $P \vdash \Gamma$, then either $P$ is in canonical form, or there exists a process $Q$ such that $P \Longrightarrow Q$.*

*Proof.* We consider the maximum prefix of cuts and mixes of $P$ such that

$$P \equiv (\nu x_1)\dots(\nu x_n)(P_1 \mid \dots \mid P_{n+m+1}),$$

and no $P_i$ is a cut. If any process $P_i$ is a link, we reduce by $(\leftrightarrow)$. If any two processes $P_i$ and $P_j$ are acting on the same channel $x_i$, we rewrite by $\equiv$ and reduce by the appropriate $\beta$-rule. Otherwise, $P$ is in canonical form. □

**Theorem 3.12** (Termination)**.** *If $P \vdash \mathcal{G}$, then there are no infinite $\Longrightarrow$-reduction sequences.*

*Proof.* As Theorem 2.14. □

# 4 Relation between CP and HCP$^-$

In this section, we discuss the relationship between CP and HCP$^-$. We prove two important theorems: every CP process is an HCP$^-$ process; and HCP$^-$ supports the same protocols as CP. We define a translation from terms in CP to terms in HCP$^-$ which breaks down the term constructs in CP into their more atomic constructs in HCP$^-$.

**Definition 4.1.**

$$\begin{array}{lcllcl}
[\![x{\leftrightarrow}y]\!] & := & x{\leftrightarrow}y & [\![(\nu x)(P \mid Q)]\!] & := & (\nu x)([\![P]\!] \mid [\![Q]\!]) \\
[\![x[y].(P \mid Q)]\!] & := & x[y].([\![P]\!] \mid [\![Q]\!]) & [\![x(y).P]\!] & := & x(y).[\![P]\!] \\
[\![x[].0]\!] & := & x[].0 & [\![x().P]\!] & := & x().[\![P]\!] \\
[\![x \triangleleft \texttt{inl}.P]\!] & := & x \triangleleft \texttt{inl}.[\![P]\!] & [\![x \triangleleft \texttt{inr}.P]\!] & := & x \triangleleft \texttt{inr}.[\![P]\!] \\
[\![x \triangleright \{\texttt{inl}: P; \texttt{inr}: Q\}]\!] & := & x \triangleright \{\texttt{inl}: [\![P]\!]; \texttt{inr}: [\![Q]\!]\} & [\![x \triangleright \{\}]\!] & := & x \triangleright \{\}
\end{array}$$

We use this relation in the first proof, and its analogue for derivations in the second.

## 4.1 Every CP process is an HCP$^-$ process

First, we prove that each CP process can be translated by this trivial translation to an HCP$^-$ process, and that this translation respects structural congruence and reduction. Reductions from CP can be trivially translated to reductions in HCP$^-$.

**Theorem 4.2.** *If $P \vdash \Gamma$ in CP, then $[\![P]\!] \vdash \Gamma$ in HCP$^-$.*

*Proof.* By induction on the derivation of $P \vdash \Gamma$. We show the interesting cases:

- Case Cut. We rewrite as follows:

$$\dfrac{P \vdash \Gamma, x : A \qquad Q \vdash \Delta, x : A^\perp}{(\nu x)(P \mid Q) \vdash \Gamma, \Delta}\ \textsc{Cut} \quad \Rightarrow \quad \dfrac{\dfrac{[\![P]\!] \vdash \Gamma, x : A \qquad [\![Q]\!] \vdash \Delta, x : A^\perp}{[\![P]\!] \mid [\![Q]\!] \vdash \Gamma, x : A \mid \Delta, x : A^\perp}\ \text{H-}\textsc{Mix}}{(\nu x)([\![P]\!] \mid [\![Q]\!]) \vdash \Gamma, \Delta}\ \text{H-}\textsc{Cut}$$

- Case $(\otimes)$. We rewrite as follows:

$$\dfrac{P \vdash \Gamma, y : A \qquad Q \vdash \Delta, x : B}{x[y].(P \mid Q) \vdash \Gamma, \Delta, x : A \otimes B}\ (\otimes) \quad \Rightarrow \quad \dfrac{\dfrac{[\![P]\!] \vdash \Gamma, y : A \qquad [\![Q]\!] \vdash \Delta, x : B}{[\![P]\!] \mid [\![Q]\!] \vdash \Gamma, y : A \mid \Delta, x : B}\ \text{H-}\textsc{Mix}}{x[y].([\![P]\!] \mid [\![Q]\!]) \vdash \Gamma, \Delta, x : A \otimes B}\ (\otimes)$$

- Case $(\mathbf{1})$. We rewrite as follows:

$$\dfrac{}{x[].0 \vdash x : \mathbf{1}}\,(\mathbf{1}) \quad \Rightarrow \quad \dfrac{\dfrac{}{0 \vdash \varnothing}\,\text{H-MIX}_0}{x[].0 \vdash x : \mathbf{1}}\,(\mathbf{1})$$

□

**Theorem 4.3.** *If $P \equiv Q$ in CP, then $[\![P]\!] \equiv [\![Q]\!]$ in HCP$^-$.*

*Proof.* By induction on the derivation of $P \equiv Q$. □

**Theorem 4.4.** *If $P \Longrightarrow Q$ in CP, then $[\![P]\!] \Longrightarrow [\![Q]\!]$ in HCP$^-$.*

*Proof.* By induction on the the derivation of $P \Longrightarrow Q$. □

**Theorem 4.5.** *If $[\![P]\!] \Longrightarrow R$ in HCP$^-$, then there is a $Q$ such that $P \Longrightarrow Q$ in CP and $R \equiv [\![Q]\!]$ in HCP$^-$.*

*Proof.* By induction on the derivation of $[\![P]\!] \Longrightarrow R$. □

## 4.2 HCP$^-$ supports the same communication protocols as CP

In this section, we prove that HCP$^-$ supports the same communication protocols as CP. This is the same as saying that it inhabits the same session types, or that the associated logical systems derive the same theorems. We show this by proving that we can internalise the hyper-environments as formulas in the logic. This is a standard method for proving the soundness of a hypersequent calculus.

We start off by defining a relation on derivations of HCP$^-$, which we call "disentanglement". This relation allows us to move applications of H-MIX downwards in the proof tree. We can use this relation to rewrite any derivation to a form in which all mixes are either attached to their respective cuts or tensors, or at the top-level.

**Definition 4.6.** *Disentanglement is the smallest relation $\rightsquigarrow$ on processes closed under the rules in Figure 1, plus the structural congruence $\equiv$. The relation $\rightsquigarrow^\star$ is the reflexive, transitive closure of $\rightsquigarrow$.*

We named this relation "disentanglement" to reflect the intuition that proof in HCP$^-$ represent multiple entangled CP proofs, which we can disentangle.

Disentanglement is terminating, and confluent up to the associativity and commutativity of mixes.

**Lemma 4.7** (Disentangle)**.** *If $P \vdash \Gamma_1 \mid \ldots \mid \Gamma_n$ in HCP$^-$, then there exist processes $P_1, \ldots, P_n$ in CP such that $P_1 \vdash \Gamma_1, \ldots, P_n \vdash \Gamma_n$ and*

$$P \vdash \Gamma_1 \mid \ldots \mid \Gamma_n \quad \rightsquigarrow^\star \quad \dfrac{[\![P_1]\!] \vdash \Gamma_1 \quad \ldots \quad [\![P_n]\!] \vdash \Gamma_n}{([\![P_1]\!] \mid \ldots \mid [\![P_n]\!]) \vdash \Gamma_1 \mid \ldots \mid \Gamma_n}\,\text{H-MIX}$$

*Proof.* We repeatedly apply the $\rightsquigarrow$-rules to the derivation $\rho$ to move the mixes downwards. There are three cases: a) if a mix gets stuck above a cut, it forms a CP cut; b) if a mix gets stuck above a $(\otimes)$, it forms a CP $(\otimes)$; c) otherwise, it moves all the way to the bottom. All applications of $(\mathbf{1})$ are followed by an application of H-MIX$_0$, forming a CP $(\mathbf{1})$. □

An environment can be internalised as a type by collapsing it as a series of pars.

**Definition 4.8.**

$$\begin{array}{lcll} \parr(\cdot) & = & \bot & \\ \parr(x_1 : A_1, \ldots, x_n : A_n) & = & A_1 \parr \cdots \parr A_n & \text{if } n \geq 1 \end{array}$$

$$\dfrac{x \notin (\mathscr{G} \mid \mathscr{H}) \qquad \dfrac{P \vdash \mathscr{G} \mid \Gamma, x{:}\,A \mid \Delta, x{:}\,A^{\perp} \qquad Q \vdash \mathscr{H}}{(P \mid Q) \vdash \mathscr{G} \mid \mathscr{H} \mid \Gamma, x{:}\,A \mid \Delta, x{:}\,A^{\perp}}\;\textsc{H-Mix}}{(\nu x)(P \mid Q) \vdash \mathscr{G} \mid \mathscr{H} \mid \Gamma, \Delta}\;\textsc{H-Cut} \quad\leadsto\quad \dfrac{\dfrac{P \vdash \mathscr{G} \mid \Gamma, x{:}\,A \mid \Delta, x{:}\,A^{\perp} \qquad x \notin \mathscr{G}}{(\nu x)P \vdash \mathscr{G} \mid \Gamma, \Delta}\;\textsc{H-Cut} \qquad Q \vdash \mathscr{H}}{((\nu x)P \mid Q) \vdash \mathscr{G} \mid \mathscr{H} \mid \Gamma, \Delta}\;\textsc{H-Mix}$$

$$\dfrac{\dfrac{P \vdash \mathscr{G} \mid \Gamma, y{:}\,A \mid \Delta, x{:}\,B \qquad Q \vdash \mathscr{H}}{(P \mid Q) \vdash \mathscr{G} \mid \mathscr{H} \mid \Gamma, y{:}\,A \mid \Delta, x{:}\,B}\;\textsc{H-Mix}}{x[y].(P \mid Q) \vdash \mathscr{G} \mid \mathscr{H} \mid \Gamma, \Delta, x{:}\,A \otimes B}\;(\otimes) \quad\leadsto\quad \dfrac{\dfrac{P \vdash \mathscr{G} \mid \Gamma, y{:}\,A \mid \Delta, x{:}\,B}{x[y].P \vdash \mathscr{G} \mid \Gamma, \Delta, x{:}\,A \otimes B}\;(\otimes) \qquad Q \vdash \mathscr{H}}{(x[y].P \mid Q) \vdash \mathscr{G} \mid \mathscr{H} \mid \Gamma, \Delta, x{:}\,A \otimes B}\;\textsc{H-Mix}$$

$$\dfrac{\dfrac{P \vdash \mathscr{G} \mid \Gamma, y{:}\,A, x{:}\,B \qquad Q \vdash \mathscr{H}}{(P \mid Q) \vdash \mathscr{G} \mid \mathscr{H} \mid \Gamma, y{:}\,A, x{:}\,B}\;\textsc{H-Mix}}{(x(y).(P \mid Q)) \vdash \mathscr{G} \mid \mathscr{H} \mid \Gamma, x{:}\,A \parr B}\;(\parr) \quad\leadsto\quad \dfrac{\dfrac{P \vdash \mathscr{G} \mid \Gamma, y{:}\,A, x{:}\,B}{x(y).P \vdash \mathscr{G} \mid \mathscr{H} \mid \Gamma, x{:}\,A \parr B}\;(\parr) \qquad Q \vdash \mathscr{H}}{(x(y).P \mid Q) \vdash \mathscr{G} \mid \mathscr{H} \mid \Gamma, x{:}\,A \parr B}\;\textsc{H-Mix}$$

$$\dfrac{P \vdash \mathscr{G}}{x[].P \vdash \mathscr{G} \mid x{:}\,\mathbf{1}}\;(\mathbf{1}) \quad\leadsto\quad \dfrac{\dfrac{\dfrac{}{0 \vdash \varnothing}\;\textsc{H-Mix}_0}{x[].0 \vdash x{:}\,\mathbf{1}}\;(\mathbf{1}) \qquad P \vdash \mathscr{G}}{(x[].0 \mid P) \vdash \mathscr{G} \mid x{:}\,\mathbf{1}}\;\textsc{H-Mix}$$

$$\dfrac{\dfrac{P \vdash \mathscr{G} \mid \Gamma \qquad Q \vdash \mathscr{H}}{(P \mid Q) \vdash \mathscr{G} \mid \mathscr{H} \mid \Gamma}\;\textsc{H-Mix}}{x().(P \mid Q) \vdash \mathscr{G} \mid \mathscr{H} \mid \Gamma, x{:}\,\perp}\;(\perp) \quad\leadsto\quad \dfrac{\dfrac{P \vdash \mathscr{G} \mid \Gamma}{x().P \vdash \mathscr{G} \mid \Gamma, x{:}\,\perp}\;(\perp) \qquad Q \vdash \mathscr{H}}{(x().P \mid Q) \vdash \mathscr{G} \mid \mathscr{H} \mid \Gamma, x{:}\,\perp}\;\textsc{H-Mix}$$

$$\dfrac{\dfrac{P \vdash \mathscr{G} \mid \Gamma, x{:}\,A \qquad Q \vdash \mathscr{H}}{(P \mid Q) \vdash \mathscr{G} \mid \mathscr{H} \mid \Gamma, x{:}\,A}\;\textsc{H-Mix}}{x \triangleleft \mathtt{inl}.(P \mid Q) \vdash \mathscr{G} \mid \mathscr{H} \mid \Gamma, x{:}\,A \oplus B}\;(\oplus_1) \quad\leadsto\quad \dfrac{\dfrac{P \vdash \mathscr{G} \mid \Gamma, x{:}\,A}{x \triangleleft \mathtt{inl}.P \vdash \mathscr{G} \mid \Gamma, x{:}\,A \oplus B}\;(\oplus_1) \qquad Q \vdash \mathscr{H}}{(x \triangleleft \mathtt{inl}.P \mid Q) \vdash \mathscr{G} \mid \mathscr{H} \mid \Gamma, x{:}\,A \oplus B}\;\textsc{H-Mix}$$

Figure 1: The disentanglement relation for HCP⁻.

**Lemma 4.9.** *If* $\vdash \Gamma$ *in CP, then* $\vdash \parr\Gamma$ *in CP.*

*Proof.* By repeated application of $(\parr)$. $\square$

Furthermore, a hyper-environment can be internalised as a type by collapsing it as a series of tensors, where each constituent environment is internalised using $\parr$. The empty hyper-environment $\varnothing$ is internalised as the unit of tensor.

**Definition 4.10.**

$$\begin{array}{lcll} \bigotimes(\varnothing) & = & \mathbf{1} & \\ \bigotimes(\Gamma_1 \mid \ldots \mid \Gamma_n) & = & \parr\Gamma_1 \otimes \ldots \otimes \parr\Gamma_n & \textit{if } n \geq 1 \end{array}$$

**Theorem 4.11.** *If* $\vdash \mathcal{G}$ *in HCP*$^-$*, then* $\vdash \bigotimes\mathcal{G}$ *in CP.*

*Proof.* By case analysis on the structure of the hyper-environment $\mathcal{G}$. If $\mathcal{G} = \varnothing$, we apply $(\mathbf{1})$. If $\mathcal{G} = \Gamma_1 \mid \ldots \mid \Gamma_n$, we apply Lemma 4.7 to obtain proofs of $\vdash \Gamma_1, \ldots, \vdash \Gamma_n$ in CP, then we apply Lemma 4.9 to each of those proofs to obtain proofs of $\vdash \parr\Gamma_1, \ldots, \vdash \parr\Gamma_n$, and join them using $(\otimes)$ to obtain a single proof of $\vdash \bigotimes\mathcal{G}$ in CP. $\square$

## 5 Related Work

Since its inception, linear logic has been described as the logic of concurrency [9]. Correspondences between the proof theory of linear logic and variants of the π-calculus emerged soon afterwards [1, 3], by interpreting linear propositions as types for channels. Linearity inspired also the seminal theories of linear types for the π-calculus [11] and session types [10]. Even though the two theories do not have a direct correspondence with linear logic, the link is still strong enough that session types can be encoded into linear types [7].

It took more than ten years for a formal correspondence between linear logic and (a variant of) session types to emerge, with the seminal paper by Caires and Pfenning [5]. This inspired the development of Classical Processes by Wadler [20].

The idea of using hypersequents to capture parallelism in linear logic judgements is not novel: Carbone *et al.* [6] extended the multiplicative-additive fragment of intuitionistic linear logic with hypersequents to type global descriptions of process communications known as choreographies. This work is distinct from our approach in that HCP$^-$ is based on classical linear logic and manipulates hypersequents differently: in Carbone *et al.* [6], hypersequents can be formed only when sequents share resources (*cf.*, H-MIX), and resource sharing is then tracked using an additional connection modality (which is not present in HCP$^-$). Montesi and Peressotti [18] revisited CP in the light of hypersequents to derive the first correspondence between the labelled transition system semantics on proof transformations, later refined in [12]. The type system of HCP$^-$ is that of [18] save for rule H-MIX$_0$ which is from [12]. This is difference means that [18] does not enjoy the translation to CLL that HCP$^-$ and HCP enjoy.

## 6 Conclusions and Future Work

In this paper, we introduced HCP$^-$, a variant of HCP which sits in between CP and HCP. It has reduction semantics, and does not allow for delayed actions, like CP, but registers parallelism using hypersequents, like HCP. This results in a calculus which structurally resembles HCP, but which is behaviourally much more like CP: all CP processes can be translated to HCP$^-$ processes, and this translation preserves the reduction behaviour of the process. The key insight to making this calculus work is to add a side condition to the logical rules in the type system which rules out self-locking processes—processes which act on both endpoints of a channel, *e.g.*, $x().x[].0$.

HCP$^-$ focuses on the basic features of CP, corresponding to multiplicative applicative linear logic. In the future, we intend to study the full version of HCP$^-$, which includes exponentials and quantifiers. Furthermore, we intend to study extensions to HCP$^-$ which capture more behaviours, such as recursive types [14] and access points [8]. Separately, we would like to extend the reduction semantics of HCP$^-$ to cover all of HCP by adding delayed actions.

## References


[1] Samson Abramsky (1994): *Proofs as processes*. Theoretical Computer Science 135(1), pp. 5–9, doi:10.1016/0304-3975(94)00103-0. Available at `http://dx.doi.org/10.1016/0304-3975(94)00103-0`.

[2] Arnon Avron (1991): *Hypersequents, logical consequence and intermediate logics for concurrency*. Annals of Mathematics and Artificial Intelligence 4(3-4), pp. 225–248, doi:10.1007/bf01531058. Available at `https://doi.org/10.1007/bf01531058`.

[3] G. Bellin & P.J. Scott (1994): *On the $\pi$-calculus and linear logic*. Theoretical Computer Science 135(1), pp. 11–65, doi:10.1016/0304-3975(94)00104-9. Available at `http://dx.doi.org/10.1016/0304-3975(94)00104-9`.

[4] Michele Boreale (1998): *On the expressiveness of internal mobility in name-passing calculi*. Theoretical Computer Science 195(2), pp. 205–226, doi:10.1016/s0304-3975(97)00220-x. Available at `https://doi.org/10.1016/s0304-3975(97)00220-x`.

[5] Luís Caires & Frank Pfenning (2010): *Session Types as Intuitionistic Linear Propositions*. In: CONCUR 2010 – Concurrency Theory: 21th International Conference, CONCUR 2010, Paris, France, August 31–September 3, 2010. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 222–236, doi:10.1007/978-3-642-15375-4_16. Available at `http://dx.doi.org/10.1007/978-3-642-15375-4_16`.

[6] Marco Carbone, Fabrizio Montesi & Carsten Schürmann (2018): *Choreographies, logically*. Distributed Computing 31(1), pp. 51–67, doi:10.1007/s00446-017-0295-1. Available at `https://doi.org/10.1007/s00446-017-0295-1`.

[7] Ornela Dardha, Elena Giachino & Davide Sangiorgi (2017): *Session types revisited*. Inf. Comput. 256, pp. 253–286, doi:10.1016/j.ic.2017.06.002. Available at `https://doi.org/10.1016/j.ic.2017.06.002`.

[8] Simon J. Gay & Vasco T. Vasconcelos (2009): *Linear type theory for asynchronous session types*. Journal of Functional Programming 20(01), p. 19, doi:10.1017/s0956796809990268. Available at `http://dx.doi.org/10.1017/S0956796809990268`.

[9] Jean-Yves Girard (1987): *Linear logic*. Theoretical Computer Science 50(1), pp. 1–101, doi:10.1016/0304-3975(87)90045-4. Available at `http://dx.doi.org/10.1016/0304-3975(87)90045-4`.

[10] Kohei Honda, Vasco Thudichum Vasconcelos & Makoto Kubo (1998): *Language Primitives and Type Discipline for Structured Communication-Based Programming*. In: *Programming Languages and Systems - ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings*, pp. 122–138, doi:10.1007/BFb0053567. Available at `https://doi.org/10.1007/BFb0053567`.

[11] Naoki Kobayashi, Benjamin C. Pierce & David N. Turner (1999): *Linearity and the pi-calculus*. *ACM Transactions on Programming Languages and Systems* 21(5), pp. 914–947, doi:10.1145/330249.330251. Available at `https://doi.org/10.1145/330249.330251`.

[12] Wen Kokke, Fabrizio Montesi & Marco Peressotti (2019): *Better late than never: a fully-abstract semantics for classical processes*. *PACMPL* 3(POPL), pp. 24:1–24:29, doi:10.1145/3290337. Available at `https://dl.acm.org/citation.cfm?id=3290337`.

[13] Sam Lindley & J. Garrett Morris (2015): *A Semantics for Propositions as Sessions*. In: *Programming Languages and Systems*, Springer Berlin Heidelberg, pp. 560–584, doi:10.1007/978-3-662-46669-8_23. Available at `https://doi.org/10.1007/978-3-662-46669-8_23`.

[14] Sam Lindley & J. Garrett Morris (2016): *Talking Bananas: Structural Recursion for Session Types*. In: *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, ICFP 2016, ACM, New York, NY, USA, pp. 434–447, doi:10.1145/2951913.2951921. Available at `http://doi.acm.org/10.1145/2951913.2951921`.

[15] Massimo Merro & Davide Sangiorgi (2004): *On asynchrony in name-passing calculi*. *Mathematical Structures in Computer Science* 14(5), pp. 715–767, doi:10.1017/s0960129504004323. Available at `https://doi.org/10.1017/s0960129504004323`.

[16] Robin Milner, Joachim Parrow & David Walker (1992): *A calculus of mobile processes, I*. *Information and Computation* 100(1), pp. 1–40, doi:10.1016/0890-5401(92)90008-4. Available at `http://dx.doi.org/10.1016/0890-5401(92)90008-4`.

[17] Robin Milner, Joachim Parrow & David Walker (1992): *A calculus of mobile processes, II*. *Information and Computation* 100(1), pp. 41–77, doi:10.1016/0890-5401(92)90009-5. Available at `https://doi.org/10.1016/0890-5401(92)90009-5`.

[18] Fabrizio Montesi & Marco Peressotti (2018): *Classical Transitions*. *CoRR* abs/1803.01049.

[19] Davide Sangiorgi (1996): *$\pi$-calculus, internal mobility, and agent-passing calculi*. *Theoretical Computer Science* 167(1-2), pp. 235–274, doi:10.1016/0304-3975(96)00075-8. Available at `https://doi.org/10.1016/0304-3975(96)00075-8`.

[20] Philip Wadler (2012): *Propositions As Sessions*. In: *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming*, ICFP '12, ACM, New York, NY, USA, pp. 273–286, doi:10.1145/2364527.2364568. Available at `http://doi.acm.org/10.1145/2364527.2364568`.

## Erratum

Since publishing this paper we discovered that Theorems 3.11 and 3.12 do not hold for the version of $\mathrm{HCP}^{-}$ in the published version of this paper. There are two solutions to this problem: a) introduce delayed actions as in [12]; b) remove hyper-environments from all logical rules as in [18]. For this revision, we opted for the second solution as it only requires minimal changes to Definition 3.5 (all other definitions and theorems in the paper are unchanged). Below we include the published version of the type system of $\mathrm{HCP}^{-}$ highlighting additions and ~~deletions~~ introduced by this revision.

**Definition 3.5** (Typing judgements)**.** *A typing judgement* $P \vdash \Gamma_1 \mid \ldots \mid \Gamma_n$ *denotes that the process* $P$ *consists of n independent, but potentially entangled processes, each of which communicates according to its own protocol* $\Gamma_i$*. Typing judgements can be constructed using the inference rules below.*

*Structural rules*

$$\frac{}{x{\leftrightarrow}y \vdash x : A, y : A^{\perp}}\ \textsc{Ax} \qquad \frac{P \vdash \mathcal{G} \mid \Gamma, x : A \mid \Delta, x : A^{\perp} \qquad x \notin \mathcal{G}}{(\nu x)P \vdash \mathcal{G} \mid \Gamma, \Delta}\ \textsc{H-Cut}$$

$$\frac{P \vdash \mathcal{G} \qquad Q \vdash \mathcal{H}}{P \mid Q \vdash \mathcal{G} \mid \mathcal{H}}\ \textsc{H-Mix} \qquad \frac{}{0 \vdash \varnothing}\ \textsc{H-Mix}_0$$

*Logical rules*

$$\frac{P \vdash \cancel{\mathcal{G} \mid} \Gamma, y : A \mid \Delta, x : B}{x[y].P \vdash \cancel{\mathcal{G} \mid} \Gamma, \Delta, x : A \otimes B}\ \otimes \qquad \frac{P \vdash \cancel{\mathcal{G} \mid} \Gamma, y : A, x : B}{x(y).P \vdash \cancel{\mathcal{G} \mid} \Gamma, x : A \parr B}\ (\parr)$$

$$\frac{P \vdash \cancel{\mathcal{G}} \varnothing}{x[].P \vdash \cancel{\mathcal{G} \mid} x : \mathbf{1}}\ \mathbf{1} \qquad \frac{P \vdash \cancel{\mathcal{G} \mid} \Gamma}{x().P \vdash \cancel{\mathcal{G} \mid} \Gamma, x : \perp}\ (\perp)$$

$$\frac{P \vdash \cancel{\mathcal{G} \mid} \Gamma, x : A}{x \triangleleft \mathtt{inl}.P \vdash \cancel{\mathcal{G} \mid} \Gamma, x : A \oplus B}\ (\oplus_1) \qquad \frac{P \vdash \cancel{\mathcal{G} \mid} \Gamma, x : B}{x \triangleleft \mathtt{inr}.P \vdash \cancel{\mathcal{G} \mid} \Gamma, x : A \oplus B}\ (\oplus_2)$$

$$\frac{P \vdash \Gamma, x : A \qquad Q \vdash \Gamma, x : B}{x \triangleright \{\mathtt{inl} : P; \mathtt{inr} : Q\} \vdash \Gamma, x : A \,\&\, B}\ (\&)$$

$$\textit{(no rule for } \mathbf{0}\textit{)} \qquad \frac{}{x \triangleright \{\} \vdash \Gamma, x : \top}\ (\top)$$

~~*Furthermore, each logical rule has the side condition that* $x \notin \mathcal{G}$.~~