Interconnection Networks in Session-based Logical Processes

Bernardo Toninho
Imperial College London
b.toninho@imperial.ac.uk

Nobuko Yoshida
Imperial College London
n.yoshida@imperial.ac.uk

Abstract

In multiparty session types, interconnection networks identify which roles in a session engage in direct communication. If role $p$ is connected to role $q$, then $p$ exchanges a message with $q$. In a session-based interpretation of classical linear logic (CLL), this corresponds to the composition, or cut, of dual propositions. This paper shows that well-formed interactions represented in a session-based interpretation of CLL form strictly less expressive interconnection networks than those specified in a multiparty session calculus. To achieve this, we introduce a new compositional synthesis property, dubbed partial multiparty compatibility (PMC), enabling us to build a global type denoting the interactions obtained by iterated composition of well-typed CLL processes. We show that the composition by a rule induces PMC global types, requiring circular interconnections between three participants. PMC is then used to define a new CLL multicast rule which can form general multiparty interconnections, preserving the deadlock-freedom property of CLL.

Keywords Session Types, Classical Linear Logic, the Pi-Calculus, Multiparty Session Types and Synthesis.

1. Introduction

The discovery of linear logic [12] and the early studies of its connections with concurrent processes [11,15] can be seen as the origin of a Curry-Howard correspondence for linear logic with typed interactive behaviours, which have led to the more recent developments connecting linear logic and (binary) session types [3]. The understanding of linear logic propositions as session types, proofs as concurrent processes and proof simplification as communication has produced new logically-motivated techniques for reasoning about concurrent processes [10,19,21,22,26,27], while also offering guidance to clean design and implementations for programming languages based on communication with strong safety guarantees such as protocol fidelity and deadlock freedom [19,22,23].

In linear logic-based session frameworks, processes communicate through a session channel that connects exactly two distinct subsystems typed by dual propositions: when one party sends, the other receives; when one party offers a selection, the other chooses. Sessions may be dynamically exchanged via a session name or creation by invocation of replicated servers. A combination of these features enables the modelling of complex behaviours between an arbitrary number of concurrent threads.

However, the linear typing discipline induced by linear logic enforces very strong separation properties on the network topology of communicating processes, to the extent that composition identifies a process by a single of its used linear channels, requiring all other linear channels in the composed processes to be disjoint and implemented by strictly separate processes. It is from these strong separation properties that deadlock-freedom arises in a rather simple typing discipline, at the cost of disallowing network topologies with more intricate connectedness properties.

This paper provides a fresh look at session-based logical processes, based on concepts originating in multiparty session types. Motivated by an industry need [23] to specify protocols with more than two interconnected, interacting parties, the multiparty session types framework [14] develops a methodology where types explicitly describe interconnection networks between many communicating processes. The basic idea of the framework consists of taking a global type (i.e. a global description of the multiparty interaction), from which we generate (or project) local types for each communicating party (specifying how a given party interacts with all others) and check that the process implementation of each party adheres to its local type. Once each process is typechecked locally, a group of many participants in a session can interact without deadlock, following a global type.

In a multiparty session type, interconnection networks identify which roles in a session engage in direct communication. If participant $p$ is connected to another participant $q$, then $p$ will exchange a message with $q$. Consider the following 3-party interaction specified as a global type $G$:

\[
G = p \rightarrow q : (\text{nat}). p \rightarrow r : (\text{bool}). r \rightarrow q : (\text{char}). \text{end} \tag{1}
\]

The type $G$ specifies an interaction where role $p$ sends to roles $q$ and $r$ a natural number and a boolean, respectively, followed by $r$ sending to $q$ a string, inducing the interconnection network depicted in Fig. 1a realisable in a system where each role is implemented by a separate process.

However, the topology of Fig. 1a is not realisable in linear logic-based session frameworks, only those of Fig. 1b. We sketch three processes with the behaviour ascribed by $G$, each implementing one role in the multiparty session which are to then be composed,

\[
P \vdash p q : A \quad pr : B \quad Q \vdash p q : A^\perp \quad qr : C \quad R \vdash pr : B^\perp \quad qr : C^\perp \tag{2}
\]

where $P$ is the implementation of $p$ with $pq$ for communication between $p$ and $q$, and $pr$ for communication between $p$ and $r$; $Q$ implements role $q$ using channel $pq$ (dually to $P'$) and $qr$ for communication with $r$, and so on.

While each process is individually typable in linear logic-based session type systems, no 3-way composition is typable. In essence, the multiparty-based approach of identifying processes with roles leads to richer interconnection topologies than the identification of processes with channels during composition, at the cost of requiring global types and projection to ensure deadlock-freedom.

In this work, we make precise the informal argument sketched above, first showing a negative result on the network intercon-
nectability in a session-based interpretation of Classical Linear Logic (CLL), translating it into a multiparty session calculus (MP), which then forms an conservative extension of CLL that allows for richer connection topologies. Though a vast amount of works on expressiveness of process calculi have been developed as enco-
dability in the literature, our approach differs insofar as our trans-
expressiveness of process calculi have been developed as richer connection topologies. Though a vast amount of works on expres-
sibulations). We show that well-formed interactions in CLL form
strictly less expressive interconnection networks than those of MP.

To demonstrate this result, we infer session flows from CLL pro-
cesses and use them to synthesise a global type. For this, we
introduce a new compositional synthesis property, dubbed partial
multiparty compatibility (PMC), which enables us to build a global
type that represents the interactions obtained by iterated composi-
cion of CLL processes. We show that the CLL composition rule in-
duces PMC global types without circular interconnections between
more than three participants (excluding the topology of Fig. 1a).

The translations from CLL into MP give two positive results: First,
CLL can be used to guarantee deadlock-freedom and termini-
ation of a (restricted) class of MP-processes with delegation and
shared channels, which is not normally guaranteed by MP-typing
systems. Secondly, PMC enables us to define a new multicut rule in
CLL which can form general multiparty interconnections, but pre-
serves the deadlock-freedom property of CLL processes. In contrast
with [8], our framework does not require any modifications to the
syntax of propositions or CLL processes, only requiring the intro-
duction of a multicut rule that is guided by PMC. We also do not
require projection from global types, thus preserving an essence of
the logical interpretations of session types.

We summarise the contributions of this work:

- A type preserving translation of typed interactions in a session-
based interpretation of classical linear logic without channel
passing and exponentials (basic CLL) into a single multiparty
session represented in MP ([2][5]).

- A compositional synthesis property, partial multiparty compat-
  ability (PMC) ([8][9]), which we use to show that the intercon-
  nection networks of CLL processes are less expressive than those
  of a single multiparty session in MP ([5]).

- A systematic extension of these results to full CLL (with chan-
  nel passage and exponentials), showing that full CLL may be en-
coded into several sessions in MP, still with no interconnection
  circuitry ([5][6][7]). As a consequence we obtain a deadlock-
  free typing discipline for interleaved multiparty sessions.

- An extension of CLL with multicut, based on PMC and the
developed translations, allowing us to type a range of untypable
examples in previous works on MP and its relationship with
linear logic [3], still ensuring deadlock-freedom and without
modifying the types or syntax of CLL ([5]).

We discuss related work in §9. The appendix §A lists omitted
definitions and proofs.

2. Processes, types and typing systems

This section introduces the two process calculi and typing disci-
plines used in our development: the binary session calculus CLL
([2][7]) which is typed using (a fragment of) the session type inter-
pretation of classical linear logic [7][27]; and the multiparty session
calculus MP [9][13]. To obtain our first negative result ([5]), we
remove channel passing and replication, addressing these features in
on the structure of $A$ as follows:

$$
\begin{align*}
\text{id}_{\bowtie \bullet A}(a,b) & \triangleright b(n).a(n) \cdot \text{id}_{A}(a,b) \\
\text{id}_{\circ \bullet A}(a,b) & \triangleright a(n).b(n) \cdot \text{id}_{A}(a,b) \\
\text{id}_{\bowtie \circ \bullet A}(a,b) & \triangleright b.\text{case} \{i : \{a.l_i; \text{id}_{A}(a,b)\} \subseteq I\} \\
\text{id}_{\circ \circ \bullet A}(a,b) & \triangleright a.\text{case} \{i : \{b.l_i; \text{id}_{A}(a,b)\} \subseteq I\} \\
\text{id}_{1}(a,b) & \triangleright 0 \\
\text{id}_{-1}(a,b) & \triangleright 0
\end{align*}
$$

The reduction semantics for CLL is defined by $P_1 \Rightarrow Q_1$ up to structural congruence $\equiv$ (we omit the homomorphic cases):

$$
\begin{align*}
a(\alpha).P \cdot |a(x).Q & \Rightarrow |P \cdot |Q(M/x) \\
a.l_i.\Pi |a.\text{case} \{i : \{\bullet, \Pi\} \subseteq I\} & \Rightarrow |P |\Pi (j \in I) \\
(\nu a)([a \leftrightarrow b] \cdot |P) & \Rightarrow |P |\{a\} (b \notin fn(P))
\end{align*}
$$

**Definition 2.2** (Deadlock-freedom). A closed process $P$ is deadlock-free if for all $P^*$ such that $P \Rightarrow^* P^* \neq P, P^* \equiv 0$.

**Proposition 2.1** (Deadlock-freedom in CLL). Suppose $P \vdash_{C\lambda} \Delta$, where $\Delta$ is either empty or contains $\mathbf{1}$ or $\bot$. $P$ is deadlock-free.

### 2.2 Multiparty session calculus

We introduce the MP calculus of multiparty sessions, where processes $P, Q$ use channels annotated by roles of the multiparty session in which they are used.

$$
P, Q ::= c[p](M), P \cdot |c[p]|(x) \cdot P \quad \text{Value send and receive} \\
c[p] < \ell, P \quad \text{Selection and branching} \\
|0 \cdot |P | \cdot |Q| \quad \text{Inaction, parallel, hiding} \\
c ::= y \cdot |s[p]|
$$

Role names are identified by $r, q, t$. Channels are ranged over by $s, t$. As before, we write $fn(P)$ and $bn(P)$ for the free and bound channels of $P$, respectively.

Role annotated channels $s[p]$ in MP are assigned local types, ranged over by $S, T$, denoting the behaviour that role $p$ performs on the given channel. Local types are defined by the grammar:

$$
S, T ::= p^!(\tau) \cdot T \mid p_!^!(\tau) \cdot T \\
\oplus_{\{i : T_i\} \subseteq I} \mid \&_{\Pi \{i : T_i\} \subseteq I} \mid \text{end}
$$

The local types $p^!(\tau) \cdot T$ and $p_!^!(\tau) \cdot T$ denote output and input to role $p$ of a value of type $\tau$, followed by behaviour $T$, respectively. Types $\oplus_{\Pi \{i : T_i\} \subseteq I}$ and $\&_{\{i : T_i\} \subseteq I}$ denote the emission (resp. reception) of a label $l_i$ to (resp. from) role $p$, followed by behaviour $T_i$, and denotes no further behaviour.

We define the set of roles of local type $T$, denoted by roles($T$), as the set of all roles occurring in type $T$. We define the typing system MP in Fig. 3 assigning the usage of role-annotated channels to local types. The judgment $P \vdash_{MP} \Delta ; \Psi$, where $\Delta$ is an unordered set of hypotheses of the form $s[p] : T$, denotes that $P$ uses its session channels according to the specification of $\Delta$, with data variables tracked in $\Psi$ (we often omit $\Psi$ for presentation purposes).

As before, we assume all bindings in $\Delta$ are disjoint and write $\Delta, \Delta'$ for the union of $\Delta$ and $\Delta'$ only defined when both contexts are disjoint and we use a judgment $\Psi \vdash_{MP} M : \tau$ for data values.

The type system relies on a notion of coherence, written $\text{co}($,$\Delta$), which ensures that $\Delta$ contains a local type for each role involved in interactions in $\Delta$. Moreover, coherence ensures that local types of interacting roles contain the necessary dual communication actions. (see §2.1 for a definition of coherence).

We define the dual of a binary type $T$, written $\bar{T}$ [13] as:

$$
\bar{T} \triangleq \downarrow_{\tau}(\tau); \bar{T} \oplus_{\Pi \{i : T_i\} \subseteq I} \triangleq \&_{\Pi \{i : T_i\} \subseteq I} \text{and end} \triangleq \&_{\downarrow_{T}} \text{and the dual rules for } \downarrow_{\cdot} \text{and } \&_{\cdot}
$$

The reduction semantics for MP processes are given below (omitting the structural congruence closure and homomorphic rules). They are fundamentally identical to the reduction rules of §2.1 but where we require not just the session channel to match (vsend) $\Psi \vdash M : \tau \quad P \vdash_{MP} \Delta, s[p] : T ; \Psi$ 

$$
\begin{align*}
s[p](a) ; & P \vdash_{MP} \Delta, s[p](\tau) \cdot T ; \Psi \\
\text{(recv)} \quad s[p] ; & P \vdash_{MP} \Delta, s[p] \cdot \tau ; \Psi \\
\text{(set)} \quad s[p](\tau) ; & P \vdash_{MP} \Delta, s[p] \cdot \tau ; \Psi \\
\text{(branch)} \quad P \vdash_{MP} \Delta, s[p] \cdot T_1 \cdots \cdots P_1 \vdash_{MP} \Delta, s[p] \cdot T_n \quad \text{for } \tau \vdash_{MP} \Delta, \Delta' ; \Psi \\
\end{align*}
$$

Figure 3: MP Typing Rules

but also the role assignment to be consistent:

- $s[p](a) ; P \vdash_{MP} \Delta, s[p] \cdot \tau ; \Psi$
- $s[p] ; P \vdash_{MP} \Delta, s[p] \cdot \tau ; \Psi$
- $s[p](\tau) ; P \vdash_{MP} \Delta, s[p] \cdot \tau ; \Psi$
- $P \vdash_{MP} \Delta, s[p] \cdot \tau ; \Psi$

We highlight that, in contrast to the CLL system, the typing system MP alone does not ensure deadlock-freedom.

**Fact 2.1** (Deadlock in MP). There exists a dead-locked process $P$ that is typable by the rules of Figure 3 i.e. $P \vdash_{MP} \emptyset$, does not imply that $P$ is deadlock-free.

**Proof.** Take $P = s[p](\tau) \cdot \{s[a] \cdot \tau\}$. $R = s[a] \cdot \tau$,

We now consider a mapping from CLL to MP. The following processes $Q$ and $Q'$ are idealised translations of $P$ and $P'$. Session $s[q](\tau)$ denotes channel $s$ located at participant $p$ which will interact with another participant $q$.

$$
\begin{align*}
Q & \triangleq s[q](\tau) \cdot s[q](\tau) \cdot s[q](\tau) ; \Psi \\
Q' & \triangleq s[q](\tau) \cdot s[q](\tau) \cdot s[q](\tau) ; \Psi
\end{align*}
$$

Morally, the processes $P$ and $Q$ above are similar, insofar as both send $\tau$ to a destination (resp. $a$ and role $p$), followed by an input (resp. on $b$ and from $r$), followed by an output of \text{“hello”} to the initial destination. A similar argument can be made for $P'$ and $Q'$. However, despite $P$ and $P'$ having the same types, we have that $Q_0 \vdash_{MP} s[q](\tau) \cdot r_1(\tau) \cdot p_1(\tau); \Psi$ and $Q' \vdash_{MP} s[q](\tau) \cdot r_1(\tau) \cdot p_1(\tau); \Psi$

By refining channels with role annotations within a multiparty session, MP can distinguish orderings of actions performed on different session sub-channels.
Thus, our goal is to find a precise way to not only systematically map process $P$ to process $Q$ (and $P'$ to $Q'$), but also generate the corresponding local typing environment in a type preserving way. To relate CLL with MP processes we define a mapping from session channels in CLL to role-annotated session channels in MP, such that given a cut-free process in CLL (intuitively, a single thread), we consistently map its channels to role-annotated channels in the MP system, capturing the cross-channel sequential dependencies that are not codified at the level of session types.

**Definition 3.1 (Channel to Role-Indexed Channel Mapping).** Let $P \vdash_{\text{CLL}} \Delta$ such that the typing derivation does not use the cut rule. We define a channel name to role-indexed channel name mapping $\sigma$ such that for all $b \in \mathfrak{f}(P); \sigma(a) = s[p][a]$ and $\sigma(b) = s[p'][a']$, then $p = p' \land q \neq q'$. We discard reflexive role assignments of the form $s[p][p]$. We write $P \vdash_{\text{CLL}}^\sigma \Delta$ to denote such a mapping and $c_\sigma(a)$, $p_{\sigma}(a)$ and $d_{\sigma}(a)$ to denote the channel, first (principal) and second (destination) roles in the image of $a$ in $\sigma$.

A mapping according to Def. 3.1 identifies a single-thread process of CLL with a single role implementation in MP, such that all its channels are mapped to same multiparty session channel $s$ and to the same principal role $p$, but to different destination roles. Formally, we note that $\forall a, b \in \mathfrak{f}(P); \sigma(a) = c_\sigma(b)$ and $p_{\sigma}(a) = p_{\sigma}(b)$. In the remainder of this paper, we exclude all reflexive role assignments.

Let $P \vdash_{\text{CLL}}^\sigma \Delta$. We write $\sigma(P)$ for the process obtained by renaming each free name $a$ in $P$ with $\sigma(a)$, where actions in $P$ are mapped to their corresponding actions in multiparty session processes and the forwarding construct is mapped to the renamed identity process (Def. 2.1).

Having constructed a syntactic mapping from CLL to MP processes, we must now find a way to generate the appropriate local typings for processes in the image of the translation.

**Definition 3.2 (Local Typing Generation).** Let $P \vdash_{\text{CLL}}^\sigma \Delta$. Let $c_\sigma$ and $p_{\sigma}$ to refer to the unique channel and principal role of $\sigma$, and $\rho$ to refer to the unique role and principal role of $\rho$, such that $\forall \rho \neq \rho'$. We inductively generate a local type $\Gamma$ such that $\sigma(P) \vdash_{\text{MP}} c_\rho p_{\rho}{\Gamma}$. By induction on the structure of typing of $P$, written $\Gamma(P)$, we write $\Gamma(P) = \rho_0 \vdash \varepsilon \end{array}$

\[
\begin{array}{ll}
\text{[0]} & \equiv \varepsilon \\
[a(M), P] & \equiv d_{\sigma}(a)(\varepsilon); [P]_{\sigma} \quad \text{with } M : \tau \\
[a(y), P] & \equiv d_{\sigma}(a)(\varepsilon); [P]_{\sigma} \quad \text{with } y : \tau \\
[a(l); P] & \equiv \oplus d_{\sigma}(a)(\varepsilon); [P]_{\sigma} \\
[a \rightarrow b] & \equiv \Pi d_{\sigma}(a)(\varepsilon); [P]_{\sigma} \\
[a \cdot \text{case}(l, P)] & \equiv \Pi d_{\sigma}(a)(\varepsilon); [P]_{\sigma}^l \\
[a \oplus b] & \equiv \Pi d_{\sigma}(a)(\varepsilon); [P]_{\sigma}^l \\
\end{array}
\]

Given the definition of a type preserving mapping of cut-free processes in CLL to single-thread processes in MP, we account for process composition. We define the judgement $P \vdash_{\text{CLL}}^\sigma \Delta, \Gamma$ such that $P$ is an n-ary composition of processes, $\Gamma$ is a typing context assigning role-indexed channel names to local types, $\Delta$ is a typing context assigning channel names to session types, $\sigma$ is a free name to role-indexed channel mapping (Def. 3.1) and $\rho$ is a mapping from bound names to role-indexed channels.

**Definition 3.3 (Process Composition).**

\[
\begin{array}{ll}
\text{[thread]} & \vdash \Delta, \alpha : A \quad Q_{\sigma} \vdash_{\rho'} \Delta', \alpha : A' \end{array}
\]

\[
\begin{array}{ll}
\rho' = \rho \cup (a, c_{\sigma}, p_{\sigma}(a)) & \vdash (P)_{\sigma} \vdash_{\rho'} \Delta, \alpha : A' \end{array}
\]

\[
\begin{array}{ll}
\forall \nu \in \Delta, \gamma \in \Delta', \nu \neq \gamma & \vdash (\nu, \gamma)_{\rho'} \vdash_{\rho'} \Delta, \alpha : A' \end{array}
\]

\[
\begin{array}{ll}
\nu & \equiv \nu(a, p_{\rho}(a)) & \nu(a) \neq d_{\sigma}(a), d_{\sigma}(a) = p_{\rho}(a) \\
\end{array}
\]

\[
\begin{array}{ll}
(\nu, \gamma)_{\rho'} \vdash_{\rho'} (P_{\sigma} | Q_{\sigma}) \equiv (\nu(a), \gamma(a)) \Delta, \alpha : A' \end{array}
\]
4. Partial multiparty compatibility

This section studies a compositional concurrent synthesis property, dubbed partial multiparty compatibility (PMC). As illustrated by Fact 2.1, a multiparty session type theory [4] cannot guarantee deadlock-freedom if we do not rely on either (1) a projection from a global type; or (2) a global synthesis property called multiparty compatibility [1][17]. For example, the previous counterexample can be avoided if we start from the following global type:

$$G = p \rightarrow q; (\cdot)G \mid p \rightarrow r; (\cdot)G \mid p$$

then type each process with the following local types.

$$T_q = q!; (\cdot)G \mid r; (\cdot)G \mid T_r = q!; (\cdot)G \mid r; (\cdot)G \mid$$

or we may build (synthesise) $G$ in $\tau$ from $\{T_p, T_q, T_r\}$ in $\tau$. If we start from a projectable global type or are able to synthesise a global type, the example in Fact 2.1 is no longer typable.

Given that CLL employs a binary form of composition, to prove negative results about expressiveness or define multiset rules in later sections, we extend a global synthesis condition to a binary (partial) relation. We introduce partial global types whose arrow $p \rightarrow q$ represents a global interaction which has not yet been composed with another party (e.g. it denotes the emission from $p$ to $q$, not yet composed with the reception by $q$). When we compose two partial specifications of participants $p$ and $q$ by fusing two partial global types, $p \rightarrow q$ is changed to $p \rightarrow q$, preserving the ordering of communications. When we finish composing all participants (thus reconstructing a complete global type – one without partial arrows), deadlock-freedom is guaranteed.

4.1 Partial global types and semantics

We first define a partial global type $G$, consisting of a combination of complete global types and endpoint interactions.

**Definition 4.1 (Partial Global Types).** The grammar of partial global types $(G, G', \ldots)$ is defined as:

$$G ::= \text{end} \mid p \rightarrow q; (\cdot)G \mid p \rightarrow q; \{l_i; G_i\}_{i \in I}$$

We omit $\uparrow, \downarrow, \oplus$ and $\&$ from the partial global types when clear from context or unnecessary. Given a partial global type $G$, we write $\text{roles}(G)$ to denote the set of role names occurring in $G$. We write $p \rightarrow q \in G$ if $p \rightarrow q$ occurs in $G$. Similarly for $p \rightarrow q \in G$. We say $G$ is complete if $p \rightarrow q \notin G$ for all $p, q \in \text{roles}(G)$.

The first three constructs of Def. 4.1 represent the standard global types [4][14]. Global type $p \rightarrow q; (\cdot)G$ means that participant $p$ sends a value of type $\tau$ to participant $q$, and the rest of interaction is specified by $G$. Global type $p \rightarrow q; \{l_i; G_i\}_{i \in I}$ means that participant $p$ selects label $l_i$, then $q$’s $i$-th branch will be chosen, becoming $G_i$. On the other hand, partial global types (the last four constructs) denote half of a complete global interaction. The absence of the selection and branching is in fact the contribution of the emission component of the interaction (from a local type q!(\cdot)G), whereas $p \rightarrow q; (\cdot)G$ denotes the reception component. A similar reasoning applies to selection and branching.

We introduce a standard notion of projection from global to local types, defined in terms of a merge operation for branchings [11], written $T \sqcup T'$, ensuring that if the locally observable behaviour of the local type is not independent of the chosen branch then it is identifiable via a unique choice/branching label (the merge operator is otherwise undefined) (see Appendix A).

**Definition 4.2 (Projection).** Let $G$ by a global type. The projection of $G$ for a role $p$ is defined by the function $G \mid p$ below. If no side conditions hold then projection is undefined.

$$s \rightarrow r: (\tau).G \mid p = \begin{cases} r(\tau); (G' \mid p) & \text{if } p = s \\ s(\tau); (G' \mid p) & \text{if } p = r \\ (G' \mid p) & \text{otherwise} \end{cases}$$

As an illustration of merging, consider

$$G_m = q \rightarrow p; \{l_1; p \rightarrow r; \{l_2; \text{end} \}, l_3; p \rightarrow r; \{l_3; \text{end} \}\}$$

Then we have

$$G_m \mid p = \&q; \{l_1; \oplus r; \{l_2; \text{end} \}, l_3; \oplus r; \{l_4; \text{end} \}\},$$

$$G_m \mid q = \oplus p; \{l_1; \text{end} \}, l_3; \text{end} \}, G_m \mid r = \&p; \{l_2; \text{end} \}, l_4; \text{end} \}$$

To define the semantics of local and global types, we introduce a swapping relation allowing for the permutation of independent global actions.

**Definition 4.3 (Swapping).** We define swapping, written $\sim_{sw}$ as the smallest congruence on complete global types satisfying the following where $p \neq p'$ and $q \neq q'$:

$$(ss) \quad p \rightarrow q; (\cdot)G \sim_{sw} p' \rightarrow q'; (\cdot)G \qquad (sb) \quad p \rightarrow q; (\cdot)G \rightarrow q' \rightarrow \{l_i; G_i\}_{i \in I} \sim_{sw} p' \rightarrow q' \rightarrow \{l_i; G_i\}_{i \in I}$$

$$\sim_{sw} \quad (bb) \quad p \rightarrow q; \{l_i; G_i\}_{i \in I} \sim_{sw} p' \rightarrow q' \rightarrow \{l_i; G_i\}_{i \in I}$$

Swapping extends to partial global types in the obvious way.

The operational semantics for local, global and configurations are given by labelled transition systems (LTS).

**Definition 4.4 (LTS for Local Types).** We define the syntax of labels as:

$$\ell ::= \text{pq}(\tau) \mid \text{pq}\ell(\tau) \mid \text{pq} \ell l_i \mid \text{pq} \ell b$$

We define $\cdot$ as $\text{pq}\ell(\tau) = \text{pq}\ell(\tau)$ and $\text{pq} \ell l_i = \text{pq} \ell b$ and vice-versa. The transition relation between local types $T \xrightarrow{\cdot} T'$ for role $p$ is defined by the following rules (assuming $k \in I$ in $\{\ell\}$):

$$\text{(out)} \quad q \uparrow (\tau); T \xrightarrow{\text{pq}\ell(\tau)} T \quad \text{(in)} \quad q \downarrow (\tau); T \xrightarrow{\text{pq}\ell(\tau)} T$$

$$\text{(sel)} \quad \{q; T_i\}_{i \in I} \xrightarrow{\text{pq}\ell(\tau)} \{T_i\}_{i \in I}$$

**Definition 4.5 (LTS for Global Types).** The transition relation between global types $G \xrightarrow{\ell} G'$ is defined by the following rules:

$$\text{(io)} \quad p \rightarrow q; (\cdot)G \xrightarrow{\text{pq}\ell(\tau)q(\cdot)G}$$

$$\text{(sb)} \quad p \rightarrow q; \{l_i; G_i\}_{i \in I} \xrightarrow{\text{pq}\ell(\tau)} G_k$$

$$\text{(sw)} \quad G_1 \sim_{sw} G_1' \land G_2 \sim_{sw} G_2' \rightarrow G_1 \sim_{sw} G_2 \Rightarrow G_1' \sim_{sw} G_2'$$

**Definition 4.6 (Configurations).** Given a set of roles $P$, we define a configuration as $C = \{T_p\}_{p \in P}$. The synchronous transition relation between configurations is defined as:

$$\text{(com)} \quad \{T_p\}_{p \in P} \xrightarrow{\ell} \{T_p\}_{p \in P} \xrightarrow{T_q} \{T_q\}_{q \in P}$$

We write $G \xrightarrow{\ell} G_{n+1}$ if $G \xrightarrow{\ell_1 \ell_2 \ldots \ell_n} G_{n+1}$ and $\ell \in \ell_1 \ell_2 \ldots \ell_n$ and similarly for $T$ and $C$. We define traces of global type $G_0$ as $\text{Tr}(G_0) = \{\ell \mid G_0 \xrightarrow{\ell} G_n, n \geq 0\}$. Similarly for configurations.

**Proposition 4.1 (Trace Equivalence).** Suppose $G$ is a well-formed global type and the set of participants in $G$ is $P$. Assume $C = \{T_p\}_{p \in P} = \{G(p)\}_{p \in P}$. Then $\text{Tr}(C) = \text{Tr}(G)$. 

5
4.2 Partial Multiparty Compatibility

In order to define partial multiparty compatibility, we first define multiparty compatibility for synchronous semantics, which is simpler than the one for an asynchronous semantics [11, 17].

Definition 4.7 (Synchronous Multiparty Compatibility). Configuration $C_0 = (T_0, p) \in P$ is synchronous multiparty compatible if for all $C_0 \xrightarrow{l} C = (T_p)_{p \in P}$ and $T_p \xrightarrow{l'} T_p'$, we introduce the composition of two partial global types, which is well-formed:

1. $\ell \in \text{pq}^1(\tau)$ or $\ell \notin I$, there exists $C \xrightarrow{\ell} C' \xrightarrow{l'} C''$;
2. $\ell \in \text{pq} \cup \{l\}$, there exists $C \xrightarrow{\ell} C' \xrightarrow{l'} C''$; or
3. $\ell \notin \text{pq} \cup l$, there exists $\ell_i = \text{pq} \cup l_i \subseteq C \xrightarrow{\ell_i} C'''$ where $\ell$ does not include actions from or to $p$.

One can check that the two sets of local types in $\text{pq}$ and $\ell$ satisfy synchronous multiparty compatibility.

Definition 4.8 (Deadlock-freedom). $C = (T_p)_{p \in P}$ is deadlock-free if for all $C \xrightarrow{\ell} C_1$, there exists $C' = (T_p')_{p \in P}$ such that $C_1 \xrightarrow{\ell} C'$ and $T_p' = \text{end}$ for all $p \in P$.

Theorem 4.1 (Deadlock-freedom, MC and a Global Type). The following rules are equivalent:

1. (MC) A configuration $C$ is synchronous multiparty compatible.
2. (DF) $C$ is deadlock-free.
3. (WF) There exists well-formed $G$ such that $\text{Tr}(G) = \text{Tr}(C)$.

Multiparty compatibility is a global property defined using the set of all participants (modelled by communicating automata in [11, 17]). To define a compositional (local) multiparty compatibility, we introduce the composition of two partial global types, which we dub as fusion.

Definition 4.9 (Fuse). We define the fusing of two partial global types $G_1, G_2$, written $\text{fuse}(G_1, G_2)$, inductively on the structure of $G_1$ and $G_2$, up to the swapping relation $\sim_{\omega}$:

$\text{fuse}(\text{end}, \text{end}) = \text{end}$

$\text{fuse}(p \rightsquigarrow q : \{\tau\}, G_1', G_2') = p \rightsquigarrow q : \{\tau\} \downarrow \text{fuse}(G_1', G_2')$

$\text{fuse}(p \rightsquigarrow q : \{\{i : G_i\}_{i \in I}\}, G_1', G_2') = p \rightsquigarrow q : \{\{i : G_i\}_{i \in I}\} \downarrow \text{fuse}(G_1', G_2')$

$\text{fuse}(p \rightsquigarrow q : \{\{i : G_i\}_{i \in I}\}, G_1', G_2') = p \rightsquigarrow q : \{\{i : G_i\}_{i \in I}\} \downarrow \text{fuse}(G_1', G_2')$

The last rule (which does not overlap with the previous three rules) can be extended similarly to input, branching and selection in partial and complete global types. We say fuse($G_1, G_2) = G$ is well-formed if $p \rightsquigarrow q \in G$ and $p \rightsquigarrow q \notin G$.

We can now define the main contribution of this section: a compositional notion of partial multiparty compatibility (PMC).

Definition 4.10 (Partial Multiparty Compatibility). Suppose $G_1$ and $G_2$ are partial global types. $G_1$ and $G_2$ are partial multiparty compatible if fuse($G_1, G_2)$ is well-formed.

Example 4.1 (Partial Multiparty Compatibility). (1) Consider the following partial global types between two participants which represent a deadlock:

$G_1 = p \rightsquigarrow q : \{\text{nat}\} \rightsquigarrow p : \{\text{bool}\}$

$G_2 = q \rightsquigarrow p : \{\text{bool}\} , p \rightsquigarrow q : \{\text{nat}\}$

Then $G_3 = \text{fuse}(G_1, G_2) = p \rightsquigarrow q : \{\text{nat}\} , q \rightsquigarrow p : \{\text{bool}\} , p \rightsquigarrow q : \{\text{nat}\}$ which is not well-formed since $p \rightsquigarrow q \in G_3$ but $q \not\rightsquigarrow p \in G_3$.

(2) Consider (as we mentioned in [8]) these correspond to the partial global types of Example 5.2. The respective complete global type is untypeable in [8]:

$G_1 = q \rightsquigarrow p \& \{l_1 : p \rightsquigarrow r, \{l_2 : \text{end}\}, l_3 : p \not\rightsquigarrow r, \{l_4 : \text{end}\}\}$

$G_2 = q \rightsquigarrow p \& \{l_1 : \text{end}\}$

We then have that:

$\text{fuse}(G_1, G_2) = q \rightarrow p : \{l_1 : p \not\rightsquigarrow r, \{l_2 : \text{end}\}, l_3 : p \not\rightsquigarrow r, \{l_4 : \text{end}\}\}$

$\text{fuse}(G_1, G_2, G_3) = q \rightarrow p : \{l_1 : p \not\rightsquigarrow r, \{l_2 : \text{end}\}, l_3 : p \not\rightsquigarrow r, \{l_4 : \text{end}\}\}$

Then: $\text{fuse}(G_1, G_2, G_3) = \text{fuse}(G_1, \text{fuse}(G_2, G_3))$.

Lemma 4.1. Suppose $\text{fuse}(G_1, G_j, G_k) = \{i, j, k\}$ is well-formed. Then $\text{fuse}(\text{fuse}(G_1, G_j), G_k) = \{i, j, k\}$.

Theorem 4.2 (Compositionality). Suppose $G_1, \ldots, G_n$ are partial global types. Assume for all $i, j$ such that $1 \leq i \neq j \leq n$, $G_i$ and $G_j$ are partial multiparty compatible and $G = \bigcup_i \text{fuse}(G_k)_{k \in I}$ is a complete global type. Then $G$ is well-formed.

5. CLL encoded as a single multiparty session

Having defined in §3 how to translate CLl processes to MP, we study the interconnection networks induced by CLl processes by generating the corresponding partial global types (§4). We prove a strict inclusion of interconnections of CLl into those of single MP, by fusing the partial global types into a complete global type.

Definition 5.1 (Generating Partial Global Types). Given a partial $P$ such that $P \vdash_{\omega} \Delta$, we generate its partial global type, written $#_o(T)$ with local type $T = [T]_o$ as follows:

$\text{thread}(G)$

$\text{comp}(G)$

Example 5.1 (Four Threads). Using the procedure defined above, we generate the following partial global types for Example 5.1:

$\text{thread}(G)$

$\text{comp}(G)$

We now make precise the claims in §3. We begin by formalising the notion of interconnection network as an undirected graph.

Definition 5.2 (Interconnection Network). Given a global type $G$ we generate the interconnection network graph for the roles of $G$ where the nodes of the graph are the roles of $G$ and two nodes $p, q$ have an edge between them (written as $p \leftrightarrow q$) if $p \rightarrow q$ or $q \rightarrow p$ or $p \leftrightarrow q$ or $q \rightarrow p$ occurs in $G$.
We first establish two basic properties of our framework: (1) we can always fuse the partial global types of a well-formed composition; (2) any two such partial types overlaps in at most 2 roles.

**Proposition 5.1.** Let $P \parallel^\sigma \Delta; \Gamma; G$ and $\Delta = \emptyset$ or $\Delta$ contains only 1 or $\bot$. There exists a single well-formed global type $G$ such that $G = \text{fuse}(G)$ where $\text{fuse}(G)$ denotes fusion of all partial global types in $G$.

**Theorem 5.1.** Let $P \parallel^\sigma \Delta; \Gamma; G$. For any distinct $\Delta_1, \Delta_2 \in G$ we have that $\text{roles}(\Delta_1) \cap \text{roles}(\Delta_2)$ contains at most 2 elements.

By the above results, we can immediately show the following separation result between general MP global types and those induced by CLL-validated processes.

**Theorem 5.2.** Let $P \parallel^\sigma \Delta; \Gamma; G$. Let $G = \text{fuse}(G)$. The interconnection network graph for $G$ is deadlock-free.

Finally, we establish that our encoding indeed produces a single multiparty session, that is, the fusing of all partial global types in a complete session is synonymous multiparty compatible.

**Theorem 5.3.** Let $P \parallel^\sigma \Delta; \Gamma; G$ and $\Delta = \emptyset$ or $\Delta$ contains only 1 or $\bot$. Then we have: (1) $P \rightarrow^\sigma \emptyset$: (2) $\text{fuse}(G)$ is well-formed and deadlock-free.

As a consequence of Prop. 5.1 and Theorem 5.3 the image of the translation into MP is deadlock-free.

**Corollary 5.1.** If $P \parallel^\sigma \Delta; \Gamma$ and $\Delta = \emptyset$ or $\Delta$ contains only 1 or $\bot$, then $\nu(\sigma(P))$ is well-formed wrt $\Delta$.

6. **Channel passing**

**Typing rules and translations** We integrate delegation (channel passing) into our framework. In CLL this amounts to replacing the rules for $\otimes$ and $\gamma$ of $\lambda$ with:

\[
\frac{}{P \vdash \Delta, y:A \quad Q_k \vdash \Delta', x:B} \quad \frac{}{P \vdash \Delta, y:A, x:B}
\]

Thus, $x:A \otimes B$ identifies a channel along which a process sends a fresh $y$ that is used according to type $A$, and then proceeds as $B$. The channels of the subprocesses using $x$ and $y$ must be disjoint. $A \gamma B$ denotes the dual behaviour. In MP, delegation is embodied by the following rules (types are extended in the obvious way):

\[
\frac{}{P \vdash \Delta, c:T} \quad \frac{}{P \vdash \Delta, x:T'}
\]

Unlike in CLL, MP processes delegate a session by performing a free output of an ambient session channel. Session input is dual. We assume role identifiers in different sessions are disjoint.

To construct a mapping from CLL to MP with delegation, we relax our single role per MP thread restriction due to the $\otimes$ typing rule, which splits the context into two distinct regions: one for usage by the emitted session and another for the continuation. If all channels were assigned to the same role, the resulting process would not be typeable in MP, which disallows for role implementations across parallel threads.

Thus, we consider a general *bijective* mapping from CLL to role-annotated MP channels, where actions in a cut-free process may map to actions pertaining to different MP roles. Moreover, since cut-free processes now have both bound and free as well as names, we define a mapping $\eta$ of bound names to variables in MP assigned to a single role, denoting the destination role for communication along the channel, which to maintain consistency with MP must be a different session than that used to send or receive the delegated name.

**Definition 6.1** (Name to Role-Indexed Channel Mapping). Let $P \vdash_{\text{CLL}} \Delta$ without using the cut rule. We define a name to role-indexed channel mapping of $P$ as a pair of mappings $(\sigma, \eta)$ such that: For all $x, y \in \text{fn}(P)$ such that $x \neq y$, if $\sigma(x) = s[p][q]$ and $\sigma(y) = s[p'][q']$, then $p \neq p'$ or $q \neq q'$ or $p \neq q'$. For all $x, y \in \text{bn}(P)$ such that $x \neq y$, if $\eta(x) = z[p]$ and $\eta(y) = z[p']$ then $p' \neq p$. Given $\eta(x) = x[p]$, we write $\eta(x)$ for $p$.

The role restrictions in Def. 6.1 guarantee that the renaming is bijective and that in a single thread we cannot implement dual role endpoints. In addition to Def. 6.1, we introduce a well-formedness condition to ensure that instances of the $\otimes$ rule do not induce roles being split across different threads.

**Definition 6.2.** Let $P \vdash_{\text{CLL}} \Delta$ without using the cut rule and $(\sigma, \eta)$ be a mapping viz. Def. 6.1. We say that $(\sigma, \eta)$ is well-formed if for each instance of $\otimes$ in the typing derivation of $P$:

\[
\begin{align*}
\frac{}{P_{1}, P_{2} \vdash_{\text{CLL}} \Delta_1, y:A} & & P_{1}, P_{2} \vdash_{\text{CLL}} \Delta_2, x:B \\
\overline{\text{fn}(x)} \vdash_{\text{CLL}} \Delta_1, y:A, x:B & & \frac{}{P_{1}, P_{2} \vdash_{\text{CLL}} \Delta_1, y:A, x:B}
\end{align*}
\]

For all $z \in \Delta_1$ and $z' \in \Delta_2$. Thus, we consider a general local typing context from each thread, assigning local types to each role annotated channel. In the cases of selection and branching we collect actions pertaining to the principal role assignment for each branch and combine the result with the context generation for the continuation processes (since it may contain different role assignments). To generate the local type for session output, we produce the delegated session type from the usage of $y$ in subprocess $P$ and proceed inductively, noting that there will be a binding for $y$ in the resulting context (since delegation is a free output in MP). For session input, we dualise the delegated type so that both input and output types match in MP, as needed.

**Definition 6.3** (Local Type Generation). Let $P \vdash_{\text{CLL}} \Delta$ where all free and bound names are distinct. We generate a local typing context $\Gamma$ such that $\sigma(\nu(P))$ is well-formed by induction on the structure of $P$, written $\lbrack P \rbrack_{\text{cll}}^\sigma$ (we write $\epsilon_{\sigma}(x)$, $\nu_{\sigma}(x)$ and $d_{\sigma}(x)$, where $\nu$ stands for $\sigma$ if $x \in \text{fn}(P)$ and $\eta$ otherwise; and $\epsilon$ for $c_{\sigma}(x)\lbrack P_{\sigma}(x)\rbrack$ if $x \in \text{fn}(P)$ and for $x$ otherwise; $T$ denotes a dual type of $T$):

\[
\begin{align*}
\lbrack 0 \rbrack & \vdash \emptyset \\
\lbrack y \rbrack & \vdash \emptyset \\
\lbrack P; Q \rbrack & \vdash \lbrack P \rbrack_{\text{cll}}^\sigma \sqcup \lbrack Q \rbrack_{\text{cll}}^\sigma \sqcup \lbrack \text{true} \rbrack_{\text{cll}}^\sigma \\
\lbrack x \rbrack & \vdash \emptyset \\
\lbrack x \cdot l_j \rbrack & \vdash \lbrack x \cdot l_j \rbrack_{\text{cll}}^\sigma \\
\lbrack x \cdot \text{case}(l_i : P_l) \rbrack & \vdash \lbrack x \cdot \text{case}(l_i : P_l) \rbrack_{\text{cll}}^\sigma
\end{align*}
\]

In Def. 6.3 we use an operation $s[p]; T[q] \Gamma$ defined as $\Gamma, \sigma; \tau$ if $s[p] \notin \Gamma$ and $\Gamma \neq \emptyset$. If $\Gamma = \tau$, $s[p] : T[q]$, we change $T'$ to $T$; $T'$ if $T$ is an input our operator; or, change $T'$ to $T$ if $T$ is a selection or branching (since $T$ already contains all actions of role $p$ by construction). If $T$ is empty we append end to $T$ in the relevant cases. See Appendix A.8 for a full definition.

**Proposition 6.1.** Let $P \vdash_{\text{CLL}} \Delta$ then $\sigma(\nu(P)) \vdash_{\text{MP}} \lbrack P \rbrack_{\text{cll}}^\sigma$. Where $\sigma(P)$ substitutes free names of $P$ by their role-indexed channel mappings and forwarding by the identity process, and $\nu(\eta(P))$ replaces non-binding occurrences of bound names by their multiparty session calculus actions with destination given by $\eta$.

As in Def. 6.1, we define the judgment $P \vdash_{\text{MP}} \nu_{\sigma} \Gamma$ where $(\sigma, \eta)$ is a well-formed bijective mapping. The key condition (**) in Def. 6.3 is changed to (**') below taking multiple role threads into account and bound names (the red letters highlight differences). We omit the rule for single threads, which just absorbs the $\nu_{\sigma} \Gamma$ judg-
ment and generates the corresponding local types (cf. Def 6.3).

\[
\begin{align*}
\{ \rho = \rho \cup \{ x, (c_x)(p_x) \mid d_x(x) \} \} \\
\{ c_x = c_x \mid (d_x)(x) \} \\
\forall x \in \Delta, x = \Delta : c_x(x) \Rightarrow d_x(x) \neq d_x(y) \wedge d_x(y) \neq d_x'(y)
\end{align*}
\]

The last line states that equal bound names in P and Q must denote different destination roles.

\[\text{(comp}_\rho\text{)} \]

\[P_2 \vdash \sigma_2 \Delta, \Delta ; \Gamma \quad Q_2 \vdash \sigma_2 \Delta, \Delta ; \Gamma \]

\[\forall x \in \Delta, x \neq \Delta = c_{x}(x) \Rightarrow p_{x}(x) \neq d_{x}(y) \wedge d_{x}(y) \neq p_{x}(y)
\]

We note that mapping role \( p \) to \( \Delta \) require the two processes to share two distinct channel names.

We can produce mappings \( \sigma \) distinct destination roles. Note that mapping role \( p \) to \( \Delta \) require the two processes to share two distinct channel names.

\[\text{(comp}_\rho\text{)} \]

\[P_2 \vdash \sigma_2 \Delta, \Delta ; \Gamma \quad Q_2 \vdash \sigma_2 \Delta, \Delta ; \Gamma \]

\[\forall x \in \Delta, x \neq \Delta = c_{x}(x) \Rightarrow p_{x}(x) \neq d_{x}(y) \wedge d_{x}(y) \neq p_{x}(y)
\]

The last line states that equal bound names in P and Q must denote different destination roles.

\[\text{(comp}_\rho\text{)} \]

\[P_2 \vdash \sigma_2 \Delta, \Delta ; \Gamma \quad Q_2 \vdash \sigma_2 \Delta, \Delta ; \Gamma \]

\[\forall x \in \Delta, x \neq \Delta = c_{x}(x) \Rightarrow p_{x}(x) \neq d_{x}(y) \wedge d_{x}(y) \neq p_{x}(y)
\]

We note that mapping role \( p \) to \( \Delta \) require the two processes to share two distinct channel names.

We can produce mappings \( \sigma \) distinct destination roles. Note that mapping role \( p \) to \( \Delta \) require the two processes to share two distinct channel names.

We can produce mappings \( \sigma \) distinct destination roles. Note that mapping role \( p \) to \( \Delta \) require the two processes to share two distinct channel names.
set of roles that may use the replicated service. Let \( P_L \vdash \theta \) \( \Gamma; \Delta \), with all free and bound names distinct. We generate a local typing context \( \Theta \) with \( \eta(\sigma(P_L)) \vdash \theta \) \( \Theta \) by induction on the structure of \( P_L \), written \( \|P_L\| \) (we show only the new cases):

- \( \|\sharp(x.y).P_L\| \) \( \Delta \) \( \Gamma \) \( \eta \) \( \Delta \) \( \Gamma \) \( \eta \)
- \( \|\varpi(y).P_L\| \) \( \Delta \) \( \Gamma \) \( \eta \) \( \Delta \) \( \Gamma \) \( \eta \)

The rules above are similar to those from \( 6 \) where we compute the local type for the replicated session inductively, by the usage of the bound name \( y \) in the continuation process \( P_L \). We note that in the top rule, \( d_s \) denotes a set of roles. We define the judgment \( P \vdash \sigma(n)^\rho \Gamma; \Delta; \Theta \) \( \rho \) for process composition, now with an additional composition rule to account for shared names:

\[
\text{(thread,)} \quad \frac{P \vdash \sigma(n)^\rho \Gamma; \Delta}{\|P\|_{\theta}}
\]

\[
\text{(comp,)} \quad \frac{P \vdash \sigma(n)^\rho \Gamma; \Delta \quad Q \vdash \sigma(n)^\rho \Gamma; \Delta \quad \rho \vdash \eta(\rho(\eta(P)))) \vdash \theta \quad \eta}{\|P; Q\|_{\theta}}
\]

\[
\text{(comp)} \quad \frac{P \vdash \sigma(n)^\rho \Gamma; \Delta \quad \rho \vdash \eta(\rho(\eta(P)))) \vdash \theta \quad \eta}{\|\varpi(x).P\|_{\theta}}
\]

Note that the condition of \( \text{(comp)} \) is \( \eta \) in \( \text{(comp)} \) in \( 6 \) where for \( \text{(comp)} \) we consider \( u \) instead of \( x \) in \( \eta \). Moreover, we require \( \sigma \) and \( \tau \) for channels in \( \Gamma \).

**Proposition 7.1.** If \( P \vdash \sigma(n)^\rho \Gamma; \Delta; \Theta \) then \( \eta(\eta(P))) \vdash \theta \); \( \Theta \).

**Example 7.1 (Replication).** The following processes are identical to an example of \( 6 \) that is not typable in the global process type system of \( 10 \):

\[
P \equiv !x(y).y(n).0 \vdash \tau \quad \tau \vdash x(:!x(y).y(n)) \equiv \text{!L}
\]

\[
Q \equiv !x(w).w(1).0 \vdash \tau \quad \tau \vdash x(:!x(w).w(1)) \equiv \text{!L}
\]

\[
R \equiv \varpi(x).\varpi(w).w(n).y(n).0 \vdash \tau \quad \tau \vdash x(:\varpi(x).\varpi(w).w(n)).y(n) \equiv \text{!L}
\]

We define mappings \( \sigma, \sigma_1, \sigma_2 \) and \( \eta_1, \eta_2, \eta_\tau \) such that:

\[
\eta(\sigma(P)) = \|\sigma(P)\|_{\theta}
\]

\[
\eta_1(\sigma_1(Q)) = \|\sigma_1(Q)\|_{\theta}
\]

\[
\eta_2(\sigma_2(R)) = \|\sigma_2(R)\|_{\theta}
\]

and so produce the following local typings:

\[
\#(P) = \|\text{!P}\|_{\theta}
\]

\[
\#(Q) = \|\text{!Q}\|_{\theta}
\]

\[
\#(R) = \|\text{!R}\|_{\theta}
\]

Then \( \varpi \equiv \sharp(x, y, z).P \), \( |Q | \), \( |R| \) \( \vdash \theta \); \( \eta \).

7.1 Interconnection Networks of CLL with Replication

Partial global type generation for replication is analogous to that of \( 6 \) insofar as we generate the replicated session type inductively and need to handle potentially multiple session channels in the image of the translation. We extend the syntax of partial global types with a dedicated replication construct \( \varpi \equiv q : !T(G) \), with the corresponding dual \( q \equiv \varpi : ?(T(G)) \), which fuse to the corresponding \( \varpi \rightarrow q : ?(T(G)) \) global type, which denotes that role \( q \) hosts the replicated behaviour \( T \), to be used by roles \( \varpi \) an arbitrary (finite) number of times.

**Definition 7.1 (Generating partial global types).** Given \( P \) such that \( P \vdash \sigma(n)^\rho \Gamma; \Delta \) we generate its partial global type \( \omega \) a multiparty session channel \( s \), written \( \#(s)^\omega(P)(s) \) by induction on the structure of \( P \) as follows (the top case is chosen if \( x \in \sigma, c_s(x) = s \), the bottom otherwise – we write only the new cases):

\[
\#(s)^\omega(\sharp(x.y).P(y)) \equiv \#(s)^\omega(P(y)) \quad \#(s)^\omega(\varpi(y).P(y)) \equiv \#(s)^\omega(P(y)) \quad \#(s)^\omega[q]^\omega(P(y)) \equiv \#(s)^\omega(P(y)) \quad \#(s)^\omega[s]^\omega(P(y)) \equiv \#(s)^\omega(P(y))
\]

We extend to \( \#(s)^\omega \Gamma; \Delta; \Theta \); \( \varpi \); \( \rho \); \( \eta \); \( \Theta \); \( G \) using the following extension to the definition of fuse:

\[
\text{fuse}(\varpi \rightarrow q : !T(G), \rho \rightarrow q : ?(T(G))) 
\]

\[
\text{fuse}(\varpi \rightarrow q : !T(G), \rho \rightarrow q : ?(T(G))) 
\]

**Theorem 7.1.** Let \( P \equiv (\eta(\rho(\eta(P)))) \vdash \theta \); \( \eta \) for some \( R \), sequences of names \( a \) and a non-replicated guarded process \( \rho.Q \).

8. Multicum in CLL

As discussed in \( 5 \) and made precise throughout our development, the inability to compose processes that interact using more than one channel – dubbed multicam – significantly limits the admissible interconnection network topologies. Logically, such a form of unrestricted multicum is unsound since it destroys the crucial cut elimination property. At the operational level, this results in the loss of deadlock freedom in a process interpretation of logic.

In this section we make use of partial multiparty compatibility (PMC) to develop a new multicum rule for CLL without delegation. While we allow for composed processes to share multiple dual channels, we restrict composition by requiring the induced partial global types to be fuseable (or PMC). With this restriction in place we recover the deadlock freedom property. Using the mapping of \( 3 \) identifying cut-free processes with the same role, we redefine the judgment \( P \vdash \sigma(n)^\rho \Gamma; \Delta \) as follows (where fuse(\( \Gamma_1, \Gamma_2 \)) is defined as fuse(\( (\#(s_1), \#(s_2), \#(s_3)), \cdots, \#(s_n), (T_0) \)) where \( c_{s_i}[^\rho(P)], T_i \in \Gamma_1 \cup \Gamma_2 \):

\[
P \vdash \sigma(n)^\rho \Gamma; \Delta, x_1:A_1, \ldots, x_n:A_n; \Gamma_1, \Gamma_2, Q \vdash \sigma(n)^\rho \Delta, x_1:A_1, \ldots, x_n:A_n; \Gamma_1, \Gamma_2
\]

\[
\text{fuse}(\Gamma_1, \Gamma_2) \text{ defined } \rho \rightarrow \varpi \Rightarrow \varpi \Rightarrow (\eta(\rho(\eta(P)))) \vdash \theta \); \( \Theta \)
\]

\[
(\varpi x_1, \ldots, x_n.P_L | Q_L) \equiv \rho(\eta(\eta(P)))) \Gamma \Delta, \varpi \Gamma_1, \Gamma_2
\]

where \( (\eta(\rho(\eta(P)))) \vdash \theta \); \( \Theta \) as follows (the red letters highlight differences).

\[
\rho \rightarrow \varpi \Rightarrow \varpi \Rightarrow (\eta(\eta(P))) \equiv \text{!L}
\]

\[
\text{fuse}(\Gamma_1, \Gamma_2) \text{ defined } \rho \rightarrow \varpi \Rightarrow \varpi \Rightarrow (\eta(\rho(\eta(P)))) \vdash \theta \); \( \Theta \)
\]

Unlike the previous sections, rule (Mcomp) is symmetric, matching a generalised cut rule. Beyond the basic restrictions such as lack of overlapping bound names in the two processes and that the role assignments for each of the composed \( x \), channels must be dual, we just maintain basic invariants akin to those of composition for \( 6 \) non-composed channels cannot have dual role assignments, the same role may not be split across different threads and role assignments cannot capture those of previously bound names. To enable multicum, the condition \( d_s(z) \neq d_s^\rho(y) \) is dropped.
Theorem 8.1 (Single Participant per Thread). Let \( P \parallel \rho \Delta; \Gamma; G \) and \( \Delta = \emptyset \) or \( \Delta \) contains only 1 or \( \bot \). Then we have: (1) \( P \rightarrow^* \rho \); and (2) fuse\((G)\) is well-formed and deadlock-free.

(1) above is by the fact that the calculus preserves a single multiparty session [14] § 5; (2) is by construction. The system (and Theorem 8.1) above is only defined in a setting without delegation. In the presence of delegation (and thus interleaved multiparty sessions) we can construct a well-typed (and fusible) composition of processes that deadlock. Intuitively, local types (and partial global types) fail to capture the linear cross-session dependencies, allowing us to interleave sessions in a non-deadlock free way.

However, in the presence of interleaved shared sessions (represented in our framework through replication), multicut preserves deadlock-freedom. By considering the channel mapping of \( \rho \Delta \) we can define a multicut rule that accounts for replication by generalising the judgement above to \( P \parallel \rho \Delta; \Gamma; \Delta; \Theta \) and adding the restriction to the \( \eta \) mapping from \( \rho \) to \( (\text{Mcomp}) \) rule. (1) below follows due to the calculus satisfying the well-linked property in [14] § 5; (2) is again by construction.

Theorem 8.2 (Exponential). Let \( P \parallel \rho \Delta; \Gamma; \Delta; \Theta; G \) and \( \Delta = \emptyset \) or \( \Delta \) contains only 1 or \( \bot \). Then we have: (1) if live\((P)\) then \( P \rightarrow P' \) (2) fuse\((G)\) is well-formed and deadlock-free.

Example 8.1 (Two Buyer Protocol [14]). The 2-buyer protocol aims to define the coordinated interactions of two Buyer agents seeking to buy from a Seller agent.

\[ \sigma(\text{Seller}) = s[S][B1](s); s[S][B1](32); s[S][B2](32); s[S][B2](\{ok:y(s),0\}, nok:0) \]

\[ \sigma_1(\text{Buyer}_1) = s[B1][S]("xpto"); s[B1][S](n); s[B1][B2](n/2); 0 \]

\[ \sigma_2(\text{Buyer}_2) = s[B2][S](n); s[B2][B1](n); s[B2][S] s\alpha; ok) \]

\[ \sigma_2(\text{Buyer}_2) = s[B2][S]("icl") \]

While in the previous development the processes above would not be composable, we can now compose them via multicut given that:

\[ \#_\sigma(\text{Seller}) = B1 \to S : \downarrow (\text{str}); S \to B1 \to \uparrow (\text{int}) \]

\[ \#_\sigma_1(\text{Buyer}_1) = B1 \to S : \downarrow (\text{str}); S \to B1 \to \uparrow (\text{int}) \]

\[ \#_\sigma_2(\text{Buyer}_2) = S \to B2 : \downarrow (\text{int}); B1 \to B2 : \downarrow (\text{int}) \]

Let \( G_1 = \#_\sigma(\text{Seller}) \), \( G_2 = \#_\sigma_1(\text{Buyer}_1) \) and \( G_3 = \#_\sigma_2(\text{Buyer}_2) \). We can see that:

\[ \text{fuse(fuse(G1, G2), G3)} = \text{fuse(fuse(G1, G2), G2)} \]

\[ = \text{ fuse(fuse(G2, G1), G1) } \]

\[ B1 \to S : (\text{str}); S \to B1 : (\text{int}); S \to B2 : (\text{int}); B1 \to B2 : (\text{int}) \]

\[ B2 \to S : (\text{ok}); B2 \to S : (\text{str}, \text{nok} : \text{end}) \]

9. Related and future work

The seminal work of [1] studies acyclicity of proofs in a computational interpretation of linear logic where names are used exactly once. With session types, channel names can be reused multiple times in a cyclic way (even in CLL defined in [2]), insofar as two processes may both send and receive along the same channel. This feature, combined with dynamic name creation in CLL, makes the study of interconnection networks (and deadlock-freedom in general) more challenging. The term “interconnection networks” in this paper originates from [2], which studies categorical models that enable cyclic networks corresponding to multicut rules. A similar discussion can be found in [27]. In our work, we make their informal observations precise, defining interconnection networks and proving the negative results in a session typed setting.

The works of [6] and [8] are the most related to our own. The work [6] proposed a typing system for CLL extended to multiparty session primitives. The main differences from our work are: (1) the work in [6] does not (aim to) study interconnection networks; (2) the projection-based approach in [8] relies on a special form of global types annotated by modalities, and linear logic propositions annotated by participants. Our synthesis-based approach requires no change to the syntax of processes, types, nor typing rules (except multicut) of CLL; (3) the multicut rule is applied to a complete set of processes in a multiparty session, directly using information of global types, while ours is a cut between two processes; and (4) the projection-based cut rule [8] is more limited than our synthesis-based rule. This is because a global type built by PMC soundly and completely characterises all deadlock-free traces observable from local type configurations (Theorem 4.1). For example, our system can type Example 7.2, which is untypeable in [8], noting that the traces of local types generated in Example 7.2 are deadlock-free.

Caires and Pérez [6] show that the binary session interpretation of intuitionistic linear logic can encode the behaviour of multiparty sessions (up to a typed barbed congruence). While the goals of this paper are quite different, focusing on interconnection network topologies of CLL processes, we note that their work does not contradict our negative results. The encoding of [6] consists of adding another participant (i.e. a centralised coordinator), that mediates all possible interactions between role implementations. Thus, a topology such as that of Fig. 1 is realised by disconnecting all participants in the network, adding a new (fourth) participant \( c \), and then connecting each participant solely with \( c \) (i.e. a tree topology). In this sense, their encoding does not preserve interconnection network topologies of global types, which is key in our criteria.

Type systems for progress in concurrent processes are a vast area of research. Usually such a typing system requires sophisticated information on channel usages (e.g. [15]) or causality among channels (e.g. [16]). While the main emphasis of this work is not the development of a type system for progress, our encodings provide a discipline for progress in restricted interleaved multiparty sessions. See Example 7.7 which is typable in our system but not in a typing system for progress in multiparty sessions [10]. Our development of multicut, which is achieved as a simple composition rule guided by PMC, ensures deadlock-freedom in the presence of interleaved shared sessions. We plan to investigate a notion of multicut for CLL that ensures progress of interleaved linear sessions. This feature would be handled by using global types with session channels [14] and synthesising a single global type consisting of several sessions, with the additional linearity checking from [14].

Multicut compatibility (MC) properties are studied in [11] where a global type is synthesised from communicating automata, assuming all participants are initially present. These global synthesis methods are not applicable to CLL where the typing rule for composition is binary. Investigations of partial compatibility for choreographies [17] and timers [4] would allow us to capture larger classes of connectability (with timing information) in the CLL framework.

References


Appendix

The structure of this appendix is as follows:

- §A.1 contains proofs and additional definitions
- §A.2 contains proofs of results from §A
- §A.3 contains proofs of results from §A
- §A.4 contains proofs of results (and additional theorems) from §A
- §A.5 contains proofs of results (and additional theorems) from §A
- §A.6 contains proofs of results (and additional theorems) from §A
- §A.7 contains proofs of results from §A
- §A.8 contains the precise definition of the ⊔ function, used throughout our development, that combines a local type with a typing environment.
- §A.9 lists the full definition of partial global type generation for CLL with delegation.
- §B lists the extended ⊨ judgement used in Section 6.
- §C gives the multicut rule for replication.
- Finally, §D gives a detailed presentation (and corresponding proofs) of an alternate mapping from CLL into MP where a thread may implement multiple roles. We prove that such a renaming does not add to the expressiveness of the topologies of the resulting interconnection network.

A. Proofs and Additional Definitions

A.1 Explanation of typing rules and reduction semantics

This section explains the typing rules and semantics for the CLL and MP calculi. In the latter case, we also include some additional definitions that are needed to precisely define the typing system, namely partial projection, merging of local and binary types and coherence.

A.1.1 The CLL System

Recalling that the rules of Figure 2 define the judgement $P_i \vdash_{CLL} \Delta; \Psi$, defined modulo the structural congruence relation $\equiv$ of Figure 4 where $\Delta$ is a set of assigning distinct channel names of $P_i$ to session types, we give an explanation of each rule: Rule (1) types the inactive process with an arbitrary session channel assigned type 1; rule (2) types the dual behaviour, which just discards the no longer used name; rule (3) accounts for the value output behaviour, typing a channel $a$ with $\tau \otimes A$ if the process outputs along $a$ a value $M$ of type $\tau$ and then proceeds by using $a$ as $A$; dually, rule (4) types a channel $a$ with $\tau \otimes \neg A$ if the process performs an input on $a$ of a value of type $\tau$ and then uses $a$ according to type $A$ in $\neg A$; we generalise both $\otimes$ and $\otimes \neg$ to emission and reception of channels, respectively, with the expected multiplicative decomposition of the context for $\otimes$; rule (5) types channel $\gamma$ with $\otimes \{l_i : a_i\}_{i \in I}$ by having the process emit a label $l_j$ with $j \in I$, and then using the channel $a$ according to the type $A_j$ in the corresponding branch; dually, rule (6) types processes that wait for a choice on channel $a$, with type $\& \{l_i : a_i\}_{i \in I}$, if the process is prepared to account for any of the possible choice labels and corresponding behaviours in the
Partial projection is defined in terms of a binary type merge operator: if no side condition applies then partial projection is undefined.

Reduction Semantics

The reduction semantics for CLL processes is given in Fig. 4 including the rules for delegation (§ 8) and replication (§ 7). We omit the standard compatible closure cases. The reduction rules for delegation and replication entail the generation of a fresh name y.

A.1.2 The MP System

To precisely define the typing system for the MP calculus, we require the following auxiliary definitions.

Definition A.1 (Partial Projection). Given a local type T, we define its partial projection onto a participant p, written T ⊥ p by induction on the structure of T according to the rules of Fig. 5. In all cases, if no side condition applies then partial projection is undefined.

Partial projection is defined in terms of a binary type merge operation for selections, where the merge T ⊔ T' of T and T' is defined by T ⊔ T' ≜ T; and with T = ⊔ {l_i : T_i} ∈ I and T' = ⊔ {l'_j : T'_j} ∈ J, T ⊔ T' = ⊔ {l_h : T_h} ∈ H, where H = H I ∪ H J ∪ H I ∩ J if l_h = l'_h for each h ∈ I \ J; and homomorphic for other types (i.e. T[T_1] ⊔ T'[T'_2] = T[T_1 \ T_2] where T is a context of local types). T ⊔ T' is undefined otherwise.

Coherence, written co(Δ), ensures that Δ contains a local type for each role involved in interactions in Δ. Moreover, coherence ensures that local types of interacting roles contain the necessary dual communication actions.

Definition A.2 (Coherence). We say Δ is coherent (denoted by co(Δ)) iff for all s[p]|T_1, there exists s[q]|T_2 ∈ Δ such that T_1 = T_2[p] and if s[p]|T_1 is in Δ, then for all q ∈ roles(T), s[q]|T_2 ∈ Δ.

The typing rules for the MP calculus are given in Figure 4, defining the judgement P ⊢_MP Δ; Ψ. Rule (vsend) types the emission of M of type τ from role p to role q assigning s[p] to the local type q ↑ (τ); T, provided the continuation P uses s[p] according to type T. Dually, rule (vrecv) types the reception of a value of type τ, bound to x in the continuation P, received by role p and sent by q, with type q ↓ (τ); T, provided P uses s[q] according to T. Rules (sel) and (branch) are the MP counterparts of rules (⊔) and (κ) from CLL (Fig. 5), respectively, with the former typing the emission a label from p to q and the latter typing the reception of a label by p from q. Rule (end) types the inactive process in a session context containing only terminated sessions. Rule (comb) types parallel composition of processes with disjoint session contexts Δ and Δ'. Finally, rule (close) types a complete multiparty session s by hiding the session channel, provided that process P uses s[1]:T_1, ..., s[n]:T_n and the corresponding role indices and local types form a coherent typing context (Def. A.2).

The following definition is used in § 4 to define the projection of a global type onto a participant.

Definition A.3 (Merge between Local Types). The merge T ⊔ T' of T and T' is defined by T ⊔ T' ≜ T; and with T = κ(τ | l_i : T_i) ∈ I and T' = κ(τ | l'_j : T'_j) ∈ J, T ⊔ T' = κ(τ | l_h : T_h) ∈ H, if l_h = l'_h for each h ∈ I \ J; and homomorphic for other types (i.e. T[T_1] ⊔ T'[T'_2] = T[T_1 \ T_2] where T is a context of local types). T ⊔ T' is undefined otherwise.

Reduction Semantics

The semantics of MP processes are given in Fig. 6 including the reduction rules for delegation (§ 8) and replication (§ 7), but omitting the standard structural congruence rules and compatible closure cases.

A.1.3 Proof of Proposition A.1

Proof. By the results in [27].

A.2 Proofs from §3: Relating the CLL and MP systems

A.2.1 Proof of Proposition A.1

Proof. The prefix case is straightforward by Definition 3.1 and (thread) in Definition 3.3, and the parallel composition uses
Since, Proof.

By Lemma 4.1, we need only show the case where $\ell \to C'$ contains only the action from/to the participant p. Hence at $C'$, we have $T_1r \to T'_{k+1}$, and $C' \to C''$ which does not include the action from/to the participant. From $C''$, we have $T_1r \to T'_{k+1}$, and $C'' \to C''$. This matches with Definition 4.1(1). The case of the input $\ell' = rq_1(\pi)$ is similar, and matches with Definition 4.1(2). In the case $\ell' = q$ is a branching such that $\ell' = rq \lor l$ which can reach $C''$ which only contains endpoints. Hence, in the case $\ell'' = rq \lor l$ such that $T_1r \to T'_{k+1}$, and $C' \to C''$. This matches with Definition 4.1(3).

(SC)⇒(WF) By (1).

(WF)⇒(DF) By definition of projection.

A.3.3 Proof of Lemma 4.1

Proof. Mechanical by induction on $G_i$, $G_j$ and $G_k$.

A.4.2 Proof of Theorem 5.1

Proof. We proceed by induction on the derivation $P \Downarrow A\Delta; \Gamma; G \quad (\text{comp}-G)$:

$\quad P_1 \Downarrow A\Delta; a; A \quad Q_k \Downarrow A\Delta, a; A; \Gamma; G \quad (\ast) \quad \text{in (comp)-}G$

By the i.h. we have that for any $G_1, G_2 \in G$, roles $\langle G_1 \rangle \cap \text{roles}(G_2)$ contains at most 2 elements. By construction we know that roles in $G$ must either appear in $\sigma'$ (corresponding to role assignments to channels in $\Delta'$ and $a$) or $\rho$ (corresponding to role assignments to bound names).

By inversion we know that $\forall z \in \Delta, y \in \Delta', d_\sigma(z) \neq d_\rho(y)$, thus there are no common $d_\sigma$ role assignments between $\sigma$ and $\sigma'$ to free names of the two processes beyond those for $a$. We also know that $p_\tau(a) = d_\rho(a)$ and $d_\rho(a) = p_\tau(a)$. By the definition of $\#_c(T_0)$ there are at least two common role names with each endpoint interaction in $G_1$ from $\sigma$ and $\sigma'$ (i.e. role assignments to free names), which are $p_\tau$ and $p_\rho$. Since $p_\tau$ is invariant and $\forall z \in \Delta, y \in \Delta', d_\sigma(z) \neq d_\rho(y)$, we have that free names in $\Delta$ cannot share any additional roles. We need now only consider $\rho$. By construction, we know that $\forall z \in \Delta, y \in \Delta', d_\sigma(z) \neq d_\rho(y)$, thus $d_\sigma(z)$ cannot appear in $G_1$ due to $\rho$. The only remaining possibilities are $p_\tau$ and $d_\rho$, which are already accounted for from the argument above. Thus we preserve the invariant and conclude the proof.

A.4.3 Proof of Theorem 5.2

Proof. Each endpoint interaction sequence in $G$ denotes the contribution of a single endpoint role in the global conversation. By Theorem 4.1 we have that any distinct pair of partial global types in $G$ shares at most 2 role names. This means that for any distinct roles $p, q, r$, if $p \leftrightarrow q \in G$ and $p \leftrightarrow r \in G$ then neither $q \rightarrow r$ nor $r \rightarrow q$ or $q \leftarrow r \rightarrow q$ in $G$. Hence, in the connection graph of $G$ we know that we cannot have triangles of the form $(p, q), (p, r), (r, q)$ as edges.

We can then see that no cycles can be formed through a “diamond” – a sequence of edges of the form $(p, q), (p, r), (r, q), (q, t)$ – in the graph by the fact that at each composition step, processes can only share one free name ($\Delta \cup \Delta' = \emptyset$) and role assignments ($\forall z \in \Delta, y \in \Delta', d_\sigma(z) \neq d_\rho(y)$, similarly for $\rho$). If we could form a “diamond” cycle in the graph, then either we have to be able to eventually compose processes sharing more than one name or with different names mapping to the same roles (to close the “diamond”).

A.4.4 Auxiliary Lemmas for Theorem 5.3

Lemma A.1. Let $P \Downarrow A\Delta; \Gamma; G \quad \text{co}(\Gamma) \implies \Delta = \emptyset$ or $\Delta$ contains only $1$ or $\bot$ and $\sigma = \emptyset$.

Proof. Assume to the contrary that $\Delta \neq \emptyset$ and doesn’t contain only $1$ or $\bot$, or $\sigma \neq \emptyset$. Then it must be the case that $P$ has some free name $x:A \in \Delta$ where $\sigma(x) = s[p][q]$, for some $s, p, q$ with $A \neq 1$ or $\bot$. By construction it must necessarily be the case that $s[p]:T \in \Gamma$. Since $x$ is free in $P$, we cannot have $s[q]:T' \in \Gamma'$ with $T'[q] = T[p]:\text{single thread } \sigma$-renamings are invariant on the $p$ role name, so for $s[q]$ to occur in $\Gamma$ it must’ve arose due to composition on $x$, which is impossible since $x$ is free, or on some other (now) bound name $y$ that mapped to $s[q][y]$, for some $r$. However, since $\sigma(x) = s[p][q]$, by construction we know that $q \neq p$. This is
Lemma A.2. Let \( P \vdash^n_\rho \Delta; \Gamma; G \) with \( \Delta = \emptyset \) or \( \Delta \) containing only \( 1 \) or \( \bot \). We have that \( \text{co}(\Gamma) \).

Proof. Assume to the contrary that \( \Gamma \) is not coherent. So (1) there exists \( s[p]:T, s[q]:T' \in \Gamma \) such that \( T[q] \not\subseteq T[p] \) or (2) \( s[p]:T \in \Gamma \) such that \( q \in \text{reset}(T) \) and \( s[q]:T' \not\in \Gamma \).

For (1) to be the case, either \( T[q] \) contains an action unmatched by \( T'[p] \) or vice-versa. Assume wlog that \( T[q] \) contains an unmatched action. Since both \( s[p]:T \in \Gamma \) and \( s[q]:T' \in \Gamma \) we know that there exists \((x, s[p][q]) \in p \) where some subprocess of \( P \) uses \( x:A \) for some \( A \) and some other subprocess of \( P \) uses \( x:A^+ \). The duality of \( x:A \) and \( \exists x:A^+ \) contradicts the existence of an unmatched action in \( T[q] \). The argument for an unmatched action in \( T'[p] \) is identical. For (2) to be the case, since \( s[p]:T \in \Gamma \) and \( s[q]:T' \not\in \Gamma \), we know that there exists \((x, s[p][q]) \in p \), with \( r \neq q \), and that \((z, s[q][z]) \not\in p \). Since \( q \in \text{roles}(T) \) then it must be the case that \( P \) uses a channel \( y \) mapped to \( s[p][q] \) that is free, which contradicts our assumptions.

A.4.5 Proof of Theorem 5.3


A.4.6 Proof of Corollary 5.1

Proof. By Proposition 3.1 and Theorem 5.3.

A.5 Proofs from §6 - Encoding Delegation

For the results in this section we extend coherence in the natural way, requiring an output of a channel of type \( T \) to match with the appropriate input of a channel of type \( T \).

A.5.1 Proof of Propositions 6.1 and 6.2

Proof. Mechanically by induction on \( P \).

A.5.2 Auxiliary Lemmas for Theorem 6.1

Lemma A.3. Let \( P \vdash^n_\rho \Delta; \Gamma; G \), \( \text{co}(\Gamma) \) implies \( \Delta = \emptyset \) or \( \Delta \) containing only \( 1 \) or \( \bot \).

Proof. Identical to Lemma D.1 noting that the renamings ensure that the two endpoints of an interaction cannot be implemented by the same single-threaded process and that bound-names involved in delegation denote linear interactions along different session channels.

Lemma A.4. Let \( P \vdash^n_\rho \Delta; \Gamma; G \) with \( \Delta = \emptyset \) or \( \Delta \) containing only \( 1 \) or \( \bot \). We have that \( \text{co}(\Gamma) \).

Proof. Identical to Lemma D.2 noting that in the case of delegation the sent channel has a dual behaviour to the received channel (but generate compatible endpoint types in \( \Gamma \)).

Proposition A.1. Let \( P \vdash^n_\rho \Delta; \Gamma; G \) and \( \Delta = \emptyset \) or \( \Delta \) contains only \( 1 \) or \( \bot \). There exists a single well-formed global type \( G \) for each session \( s \), such that \( G = \text{fuse}(\Gamma) \) where \( \text{fuse}(\Gamma) \) denotes fusion of all partial global types for session \( s \) in \( G \).

Proof. Since \( \Delta \) is empty or contains only \( 1 \) or \( \bot \) we have that \( P \) is an \( n \)-ary composition of (cut-free) processes. If \( n = 1 \) then \( P = 0 \) and its corresponding global type is just \( \text{end} \). The interesting case is when \( n > 1 \). Since the context is either empty or contains only \( 1 \) or \( \bot \), we have that \( P \) is of the form \( \nu a \)(\( P_1 \) \( \cdots \) \( P_n \)) where all free names \( a;A_i \) of each of the \( P_i \) processes are cut with some other \( P_i \) using \( a_j;A_j^- \). Thus, by construction of \( P \) we have that for each bound name \( a \) of \( P \) we have \( c_p(a)[p \rho(a)]:T \in \Gamma \) and \( c_p(a)[d_p(a)]:T' \in \Gamma \) with \( T \) \( d_p(a) = T' \) \( \rho_p(a) \) and thus for each action between two roles in a partial global type in \( \Gamma \) in \( G \) we can always find a matching action in another partial global type in \( G \); therefore we can fuse all partial global types in \( G \) into a single global type.

Lemma A.5. Let \( P \vdash^n_\rho \Delta; \Gamma; G \) and \( \Delta = \emptyset \) or \( \Delta \) containing only \( 1 \) or \( \bot \). The interconnection network graph for \( \text{fuse}(\Gamma) \) for each session is acyclic.

Proof. The proof is identical to that of Theorem D.1. We observe that delegation does not add to the connection graph since delegation denotes a distinct multiparty session (in fact, a binary session).

We note that forwarding combined with delegation also do not add to the connection graph, since a forwarder just “ping-pongs” between two channels. For instance in \( x(y)[y \leftrightarrow x] \), the forwarder is implemented by a process that will perform an action on \( y \) and a dual action on \( x \). Thus if \( x \) is mapped to \( s[p][q] \) and \( y \) instantiated with \( s'[r][t] \) we already had an edge between \( p \) and \( q \) in the connection graph due to \( x \) and one between \( r \) and \( t \) (in a graph for \( s' \)) due to \( y \).

Similarly, in \( \exists x \exists y \sum (z \leftrightarrow y) \wedge P \), the forwarder simply ping pongs between \( z \) and \( y \), mediating between the process implementing a session on \( z \) with the process receiving \( y \).

A.5.3 Proof of Theorem 6.1

Proof. (1) follows from [19, 27]. (2) follows from Prop. A.1. (3) follows from Lemma A.3.

A.6 Proofs from §7 - Encoding Replication

For the results in this section we extend coherence in the natural way, requiring a replicated input to match with the appropriate replicated output.

A.6.1 Proof of Proposition 7.1

Proof. Mechanical by induction on \( P \) (similar with Propositions 6.1 and 6.2).

A.6.2 Auxiliary Lemmas and Definitions for Theorem 7.1

Lemma A.6. Let \( P \vdash^n_\rho \Delta; \Theta; G \), \( \text{co}(\Theta) \) implies \( \Delta = \emptyset \) or \( \Delta \) contains only \( 1 \) or \( \bot \) and \( \sigma = \emptyset \).

Proof. As Lemma A.3

Lemma A.7. Let \( P \vdash^n_\rho \Delta; \Theta; G \) with \( \Delta = \emptyset \) or \( \Delta \) containing only \( 1 \) or \( \bot \). We have that \( \text{co}(\Theta) \).

Proof. As Lemma A.3

Theorem A.1. Let \( P \vdash^n_\rho \Delta; \Theta; G \), \( \text{co}(\Theta) \) iff \( \Delta = \emptyset \) or \( \Delta \) contains only \( 1 \) or \( \bot \) and \( \sigma = \emptyset \).

Proof. From Lemmas A.6 and A.7

Extensions to Exponential in MP The system which contains exponential as sessions is studied in [20]. We extend several definitions for the exponential constructs based on [19] as follows. Note that [20] uses \( ! \) for \( \Theta \) and \( ? \) for \( \Theta \) in our paper.

1. The syntax of \( T \) includes \( \overline{p}(T) \) and \( \overline{p}(T) \).
2. The duality of binary types is extended to \( \overline{!}(T) = ! ?(T) \) and \( ?(T) = ! !(T) \).
3. A projection of $T$ into the binary types are defined as an input and an output.
4. $G = \delta \rightarrow r \cdot \tau r(T).end$ is projected to
   - $G|p = r'(T) if p ∈ \delta$;
   - $G|p = \emptyset(T) if p = r$;
   - $G|p = (G'|p)$ otherwise.

Note that the projection from the global types fused from the translations is always defined and $*$ does not have a continuation.

5. The labels for LTSs are extended with $pq(T)$ and $pq?(T)$, and the LTS rules of the local types are defined as:
   
   \( pq(T) \xrightarrow{\text{p}} T \xrightarrow{\text{q}} G(T) \) \hspace{1cm} \text{(bang)} \hspace{1cm} \text{\textbar} \text{T} \xrightarrow{\text{q}(T)} T(T) \)

   We extend the duality of labels as $pq(T) = q.p(T)$ and $q.p?(T) = pq?(T)$ with $q.p$. Then the transitions of the configurations do not change.

6. We extend Definition 4.7 by adding the following cases.
   - if $\ell = pq?$(T) there exists $C \xrightarrow{l} C'$ such that $C'' = (T_p)_{p \in P}$, $T_p \xrightarrow{l} T_p'$.
   - if $\ell = pq?(T)$ for all $\ell \xrightarrow{C} C''$ such that $C'' = (T_p)_{p \in P}$, $T_p \xrightarrow{l} T_p'$.

   Theorem 4.1 is updated replacing (DF) by the following liveliness property:

\begin{definition}[Live] $G = (T_{(p)}_{p \in P})$ is live if for all $C \xrightarrow{l} C_l = T'_{(p)} \xrightarrow{l} T'_{(p)} \not\equiv l$, there exists $C'' = (T'_{(p)}_{p \in P})$ such that $C_1 \xrightarrow{l} C''$ and $C'' \xrightarrow{l} C''$ if $l$ is an output or a selection; or (2) $C' \xrightarrow{l} C''$ if $l$ is an input; or (3) $C'' \xrightarrow{l} C''$ if $l = pq \triangleright l$ with some $l' = pq \triangleright l$.
\end{definition}

Then the rest is proved as Theorem 4.1.

\begin{proposition} Let $P \overset{\nu}{\Rightarrow} (\sigma,\eta) Ω;\Delta;G \text{ and } \Delta = \emptyset$ or $\Delta$ contains only $1$ or $\bot$. There exists a single well-formed global type $G$ for each session $s$, such that $G = \text{fuse}(G)$ where $\text{fuse}(G)$ denotes fusion of all partial global types for session $s$ in $G$.
\end{proposition}

\begin{proof} Since $\Delta$ is empty or contains only $1$ or $\bot$ we have that $P$ is an $n$-ary composition of (cut-free) processes. If $n = 1$ then $P = 0$ and its corresponding global type is just $G$. The interesting case is when $n > 1$. Since the context is either empty or contains only $1$ or $\bot$, we have that $P$ is of the form $(\nu a)(P_1 \cdot \cdots \cdot P_n)$ where all free names $a;A_1$ of each of the $P_i$ processes are cut with some other $P_i$ using $a;A_1$. Thus, by construction of $\nu$ we have that for each bound name $a$ of $P$ we have $c_\sigma(a) = p_a(a)|T_\sigma \in \Gamma$ and $c_\eta(a) = \tau \cdot p_a(a)$ and thus for each action between two roles in a partial global type in $G$, we can always find a matching action in another partial global type in $G'$, therefore we can fuse all partial global types in $G'$ into a single global type.
\end{proof}

\begin{theorem} Let $P \overset{\nu}{\Rightarrow} (\sigma,\eta) Ω;\Delta;G$. If $\Delta = \emptyset$ or containing only $1$ or $\bot$, theptions are not different threads to use the same replicated channel.

\end{theorem}

\begin{proof} As before, noting that it is not possible for different threads to use the same replicated channel.

We note that even if we had a MIX-like rule which did enable for such a process to be typable, each instance of the replicated server uses a distinct session channel and thus denotes distinct global types and connection graphs.
\end{proof}

\[\begin{array}{ll}
\text{A.6.3 Proof of Theorem 7.1} & \\
\text{Proof.} (1) follows from \text{[7] 27}. (2) follows from Prop. A.2 (3) follows from Theorem A.2.
\end{array}\]

\[\begin{array}{ll}
\text{A.7 Proofs from 8 Multicut} & \\
\end{array}\]

\[\begin{array}{ll}
\text{A.7.1 Proof of Theorem 8.1} & \\
\text{Proof.} (1) By the same proposition as Proposition 8.1, we obtain if $P \overset{\nu}{\Rightarrow} (\sigma,\eta) Ω;\Delta;\Gamma$, then $\rho(\sigma(P)) \overset{\nu}{\Rightarrow} (\sigma,\eta) Ω;\Delta;\Gamma$. By the definition, $\Delta$ only contains a single multiparty session, each prefix is simple (\text{Definition 5.25 in JACM}). Since $\Delta = \emptyset$ or $\Delta$ contains only $1$ or $\bot$, $\Gamma$ is coherent. By the result of \text{[1] 5}, $\rho(\sigma(P)) \rightarrow^* 0$. Then by the operational correspondence between MP and CLL, $P \rightarrow^* 0$. (2) is by construction of (Mconv).
\end{array}\]

\[\begin{array}{ll}
\text{A.7.2 Proof of Theorem 8.2} & \\
\text{Proof.} (1) By the same reasoning as above, noting that as the image of CLL, ! and ? names (shared names below) satisfy the following well-linked condition from \text{[1] 5}(\text{Definition 5.27 in JACM}) when $\nu$-name is active:
\end{array}\]

\text{We say} $P$ \text{is well-linked when for each $P \rightarrow^* Q$, whenever $Q$ has an active prefix whose subject is a (free or bound) shared name, then it is always part of a redex.}

\end{proof}

Hence the translation into MP satisfies the liveness property. Since $P \overset{\nu}{\Rightarrow} (\sigma,\eta) Ω;\Delta;\Gamma$ then $\sigma(\eta(\rho(P))) \overset{\nu}{\Rightarrow} (\sigma,\eta) Ω;\Delta;\Gamma$ and the operational correspondence between MP and CLL, CLL satisfies the liveness.

\end{proof}

\[\begin{array}{ll}
\text{A.8 Definition of } \nu & \\
\text{Definition 5.27 in JACM}
\end{array}\]

\[\begin{array}{ll}
\text{A.9 Partial Global Type Generation for Delegation} & \\
\end{array}\]

\begin{figure}[]
\caption{List the partial global type generation for $\nu$}
\end{figure}

\[\begin{array}{ll}
\text{B. Extended } \xi \text{ judgement} & \\
\end{array}\]
that for all (Local Typing) selection and branching we collect actions pertaining to the principal local types to each role annotated channel. In the cases of second roles in the image of a mapping and \( b \in \text{Definition } \mathbb{D.1} \) may map to actions pertaining to different we consider a generalized mapping from \( \Gamma \). In a single thread we cannot implement such that \( \Gamma \) is bijective and that in a single thread we cannot implement.

\[
\begin{align*}
\#_\sigma^P(\emptyset)(s) & \triangleq \text{end} \\
\#_\sigma^P(x(y), (P_1 \mid P_2))(s) & \triangleq \begin{cases} 
\rho_\sigma(x) \leadsto d_\sigma(x) \uparrow (\langle P_1 \rangle \#_\sigma^P(y))(s), \#^P_\sigma(P_1)(s), \#^P_\sigma(P_2)(s) \\
\text{fuse}(\#^P_\sigma(P_1)(s)), \#^P_\sigma(P_2)(s)) 
\end{cases} \\
\#_\sigma^P(x.y)(P_1)(s) & \triangleq \begin{cases} 
\rho_\sigma(x) \leadsto p_\sigma(x) \downarrow (\langle P_1 \rangle \#_\sigma^P(y))(s), \#^P_\sigma(P_1)(s) \\
\#^P_\sigma(P_1)(s) 
\end{cases} \\
\#_\sigma^P(x.I_i; P_i)(s) & \triangleq \begin{cases} 
\rho_\sigma(x) \leadsto d_\sigma(x); \{\#^P_\sigma(P_i)(s)\} \\
\#^P_\sigma(P_i)(s) 
\end{cases} \\
\#_\sigma^P(x.\text{case}(I_i; P_i)_{i \in I})(s) & \triangleq \begin{cases} 
\rho_\sigma(x) \leadsto p_\sigma(x); \& \{\#^P_\sigma(P_i)(s)\}_{i \in I} \\
\#^P_\sigma(P_i)(s) 
\end{cases}
\end{align*}
\]

Figure 7: Partial Global Types for Delegation

C. Multicut for Replication

Using the name to role-indexed channel mapping of \( \mathbb{D.7} \) we redefine the judgement \( P \triangleright^\sigma_{\rho_{\sigma}} \Delta; \Gamma; \Theta \) as follows:

\[
\begin{align*}
\text{(Mcomp)} \\
P_\rho \triangleright^\sigma_{\rho_{\sigma}} \Delta; \Delta', x_1; A_1, ..., x_n; A_n; \Theta_1 \\
Q_\sigma \triangleright^\rho_{\rho_{\rho}} \Delta; \Delta', x_1; A_1, ..., x_n; A_n; \Theta_2 \\
\text{fuse}(\Theta_1, \Theta_2) \text{ defined } \rho' \cap \rho = \emptyset \quad (\diamond)
\end{align*}
\]

where \((\diamond)\) – key changes from \((\dag)\) are highlighted in red:

\[
\begin{align*}
\rho'' = \rho \cup \rho' \cup \{x_i, c_{\sigma}(x_i)[p_\sigma(x_i)[d_\sigma(x_i)]]\} \\
c_{\sigma}(\rho'') \triangleq c_{\sigma}(\rho') \cup \{x_i, c_{\sigma}(x_i)[p_\sigma(x_i)[d_\sigma(x_i)]]\} \\
\forall \gamma \in \Delta, y \in \Delta', \gamma' \triangleq \gamma_{\rho'}(y) \Rightarrow \gamma_{\sigma}(y) \triangleq \gamma'_{\sigma}(y) \\
\land \rho_{\gamma}(y) \neq \rho_{\gamma}(y) \land \rho_{\gamma'}(y) \neq \rho_{\gamma'}(y) \land \rho_{\gamma'}(y) \neq \rho_{\gamma'}(y) \\
\forall \gamma \in \eta_{\gamma}, \forall \gamma' \in \eta_{\gamma'}[x^\#_\sigma^P(y)] \Rightarrow \eta_{\gamma}(x) \neq \eta'_{\gamma}(y) \\
(\sigma'(\rho') = \sigma'(\gamma'))
\end{align*}
\]

D. CLL threads as multiorple MP threads

The results developed in our may seem to hinge on the fact that we map CLL cut-free processes to simple role processes in MP. In this section we prove this is not the case by showing that if we consider a generalized bijective mapping from CLL channels to role-annotated MP channels (i.e. where actions in a cut-free process may map to actions pertaining to different MP roles), we may reconstruct identical results.

Definition D.1 (Bijective Channel to Role-Indexed Channel Mapping). Let \( P \vdash_{\mathbb{C.L}} \Delta \) not using the cut rule. We define a bijective channel name to role-indexed channel name mapping \( \sigma \) such that for all \( a \in \text{fn}(P_\rho) \), \( \sigma(a) = s[p][q] \), for some \( s, p, q \), where \( b \in \text{fn}(P_\sigma) \) with \( b \neq a \) implies \( \sigma(b) = s[p'][q'] \) such that \( p \neq q' \land q \neq p' \), \( p \neq p' \lor q \neq q' \). We write \( P_\rho \triangleright^\sigma_{\rho_{\sigma}} \Delta \) for such a mapping and \( c_{\sigma}(a), p_\sigma(a) \) and \( d_\sigma(a) \) to denote the channel, first and second roles in the image of \( a \) in \( \sigma \).

The role restrictions in the definition guarantee that the renaming is bijective and that in a single thread we cannot implement dual role endpoints. Since threads now denote potentially multiple roles, we must generate a local typing context from each thread, assigning local types to each role annotated channel. In the cases of selection and branching we collect actions pertaining to the principal role assignment for each branch and combine the result with the context generation for the continuation processes (since it may contain different role assignments).

Definition D.2 (Local Typing). We generate a local typing context \( \Gamma \) such that \( \sigma(P_\rho) \triangleright^\sigma_{\rho_{\rho_{\sigma}}} \Gamma \) by induction on the structure of \( P_\rho \).

written \( [P]_{\sigma}(c) \) for the type binding for \( c \) in \( [P]_{\sigma} \) or end if no such binding exists and \( c \) for \( c_{\sigma}(\alpha)[p_\sigma(x)] \):

\[
\begin{align*}
[\emptyset]_{\sigma} & \triangleq \emptyset \\
[a(M).P]_{\sigma} & \triangleq c_{\sigma}(a)[\tau](\{P\})_{\sigma} \quad \text{with } M : \tau \\
[a(y).P]_{\sigma} & \triangleq c_{\sigma}(a)[\tau](\{P\})_{\sigma} \quad \text{with } y : \tau \\
[a.I_i; P_i]_{\sigma} & \triangleq c_{\sigma}(a)[\{I_i, P_i\}(c)]_{\sigma} \cup [P]_{\sigma} \\
[a.\text{case}(I_i; P_i)_{i \in I}]_{\sigma} & \triangleq c_{\sigma}(a)[\{I_i, P_i\}(c)]_{\sigma} \cup [P]_{\sigma} \\
[a \oplus b]_{\sigma} & \triangleq [d_\sigma(a), b]_{\sigma} \quad \text{with } \Delta = a : A, b : A^+ \\
\end{align*}
\]

We define the judgment \( P \triangleright^\sigma_{\rho_{\sigma}} \Delta; \Gamma \) as before, but where \( \sigma \) is a mapping according to Definition D.1 and \( \bullet \) is defined as:

\[
\begin{align*}
\rho'' = \rho \cup \{x_i, c_{\sigma}(x_i)[p_\sigma(x_i)[d_\sigma(x_i)]]\} \\
\forall \gamma \in \Delta, y \in \Delta', \gamma'' \triangleq \gamma_{\rho'}(y) \Rightarrow \gamma_{\sigma}(y) \triangleq \gamma'_{\sigma}(y) \\
\land \rho_{\gamma}(y) \neq \rho_{\gamma}(y) \land \rho_{\gamma'}(y) \neq \rho_{\gamma'}(y) \land \rho_{\gamma'}(y) \neq \rho_{\gamma'}(y) \\
\forall \gamma \in \eta_{\gamma}, \forall \gamma' \in \eta_{\gamma'}[x^\#_\sigma^P(y)] \Rightarrow \eta_{\gamma}(x) \neq \eta'_{\gamma}(y) \\
(\sigma'(\rho') = \sigma'(\gamma'))
\end{align*}
\]

Proposition D.1. If \( P \triangleright^\sigma_{\rho_{\sigma}} \Delta; \Gamma \) then \( \rho(\sigma(P)) \triangleright^\sigma_{\rho_{\rho_{\sigma}}} \Delta; \Gamma \)

D.1 CLL bijective mapping as a single MP session

We replay the development of \( \mathbb{S.3} \) for our bijective mapping.

Definition D.3 (Generating Partial Global Types). Given \( P \) such that \( P \vdash_{\mathbb{C.L}} \Delta \) we generate its partial global type, written \( \#_{\sigma}(P) \) by induction on the structure of \( P \) as follows:

\[
\begin{align*}
\#_{\sigma}(\emptyset) & \triangleq \text{end} \\
\#_{\sigma}(a(M), P) & \triangleq p_\sigma(a) \to d_\sigma(a) \uparrow (\tau); \#_{\sigma}(P) \quad \text{with } M : \tau \\
\#_{\sigma}(a(y), P) & \triangleq d_\sigma(a) \to p_\sigma(a) \downarrow (\gamma); \#_{\sigma}(P) \quad \text{with } y : \tau \\
\#_{\sigma}(a.I_i; P_i) & \triangleq p_\sigma(a) \to d_\sigma(a); \{I_i; \#_{\sigma}(P_i)\} \\
\#_{\sigma}(a.\text{case}(I_i; P_i)_{i \in I}) & \triangleq d_\sigma(a) \to p_\sigma(a); \& \{I_i; \#_{\sigma}(P_i)\}_{i \in I}
\end{align*}
\]

Exactly as before, we generalize the \( \triangleright^\sigma \) judgment with a set of partial global types, written \( P \triangleright^\sigma_{\rho_{\sigma}} \Delta; \Gamma; \Theta \). We may then replicate
the results of \(\texttt{§5} \) uniformly.

\[\text{(thread-} G)\]

\[P_1 \perp^n \Delta \quad p = 0\]

\[P_1 \llp_{\rho} \Delta \cdot \{P_2\} ; \{\#_\sigma (P_3)\}\]

\[\text{(comp-} G)\]

\[P_1 \perp^n \Delta \cdot x : A \quad \text{Q}_1 \llp_{\rho} \Delta \cdot x : A ; \Gamma ; G \quad (\bullet)\]

\[\forall z \in \Delta \cdot y : \Delta ; c(y) = c_{\sigma}(y) \implies p_\rho(y) \neq p_{\sigma}(y) \quad (\forall z \in \Delta \cdot x : \Delta \cdot c(x) = c_{\sigma}(y) \implies p_\rho(x) \neq p_{\sigma}(y))\]

\[\nu(z) (P_1 \mid Q_2) \llp_{\rho} ^{\#(\sigma (z)) \chi(x)} \Delta \cdot \Delta ; \Gamma ; [P_2]_\rho ; G \cup \{\#_\sigma (P_3)\}\]

**Proposition D.2.** Let \(P \llp_{\rho} \Delta ; \Gamma ; G\) and \(\Delta = \emptyset\) or \(\Delta\) contains only 1 or \(\bot\). There exists a single well-formed global type \(G\) such that \(G = \text{fuse}(\lambda)\) where \(\text{fuse}(\lambda)\) denotes fusion of all partial global types in \(G\).

**Proof.** Since \(\Delta\) is empty or contains only 1 or \(\bot\), we have that \(P\) is an \(n\)-ary composition of (cut-free) processes. If \(n = 1\) then \(P = 0\) and its corresponding global type is just end. The interesting case is when \(n > 1\). Since the context is either empty or contains only 1 or \(\bot\), we have that \(P\) is of the form \((\eta) (P_1 \mid \cdots \mid P_k)\) where all free names \(\alpha_1 : A_1\) of each of the \(P_i\) processes are cut with some other \(P_i\) using \(\alpha_i : A_i\). Thus, by construction of \(\llp\) we have that for each bound name \(x\) of \(P\) we have \(c_{\rho}(x)[p_\rho(a) : T \in \Gamma\) and \(\forall z \in \Delta \cdot x : \Delta \cdot c(y) = c_{\sigma}(y) \implies p_\rho(x) \neq p_{\sigma}(y))\) and thus for each action between two roles in a partial global type in \(G\) we can always find a matching action in another partial global type in \(G\), therefore we can fuse all partial global types in \(G\) into a single global type.

**Theorem D.1.** Let \(P \llp_{\rho} \Delta \cdot \Gamma ; G\). Let \(G\) be the fusion of all partial global types in \(G\) into a single global type. If \(\Delta = \emptyset\) or containing only 1 or \(\bot\). The interconnection network graph for \(G\) is acyclic.

**Proof.** Assume to the contrary that the connection graph for \(G\) has a cycle. We have two kinds of cycles: a sequence of edges of the form \(p, q, (r, q), (r, q), \ldots\) or a sequence of edges of the form \(p, q, (r, q), (r, t_1), \ldots, (t_m, s), (r, v_1), \ldots, (v_m, s)\). We show that \(\Delta\) cannot be empty or contain only 1 or \(\bot\) in either case, deriving a contradiction.

Assume the connection graph for \(G\) contains a cycle. Since \((p, q)\) is in the connection graph, we know that roles \(p\) and \(q\) cannot be implemented in the same process thread. Similarly for \(p\) and \(r\) and \(q\) and \(\rho\). Thus, we must have at least three process threads \(P_1, P_2, P_3\), one for each role. Without loss of generality, assume \(P_1 \perp^n x : A ; y : B\) with \(\sigma(x) = s[p][q]\) and \(\sigma(y) = s[p][r]\) and \(\sigma(z) = s[s][t]\) for \(\sigma(\omega) = s[s][t]\) and \(\sigma(\omega) = s[s][t]\) and \(\sigma(\omega) = s[s][t]\). This is impossible by Definition \(\text{D.1}\).

When \(n \geq 1\) and \(m = m' + 1\), we have established that \(p\) and \(q\), \(p\) and \(q\) and \(r\) and \(r\) and \(q\) cannot be implemented by the same thread. If \(q\) and \(q\) were implemented by the same thread, we can easily see that we cannot find a closed instance of this network since we’d need to compose with the (distinct) implementations of \(p\) and \(r\) which are themselves connected and thus we’d need to compose a process mapped to \(s[p][q]; s[v][r]; s[v][q]\) with a process mapped to \(s[p][q]; s[p][r]\) and another mapped to \(s[p][q]; r[v][q]\). If we compose the first with the second, we cannot compose with the third since they would share two channelrole assignment pairs: \(p[r]\) and \(s[q][v]\). A similar reasoning applies to the other ways of composing the processes. It is also easy to see that the same reasoning applies if \(q\) and \(v\), \(v\), \(v\), \(v\) were implemented by the same thread.

When \(n \geq n' + 1\) and \(m = m' + 1\) we begin with the case where \(q\) and \(v\) are implemented by the same thread and \(r\) and \(t\) are implemented by the same thread. It is easy to see we cannot compose such processes. The same reasoning applies as we increase \(n\) and \(m\). If \(t\), \(v\), or \(v\) are implemented by the same process, we must have processes sharing two distinct channels, which we cannot. If \(t\) and \(v\) are the same process, then the implementation of \(s\) must share two channels with this process, which is also a contradiction. Finally, if \(p\) and \(s\) are the same process, we are in the same situation as that described in the case in which \(n = 0\) and \(m = 0\).

**Lemma D.1.** Let \(P \llp_{\rho} \Delta ; \Delta ; \Gamma ; G\). \(\text{co}(\Delta)\) implies \(\Delta = \emptyset\) or \(\Delta\) contains only 1 or \(\bot\) and \(\sigma = \emptyset\).

**Proof.** Assume to the contrary that \(\Delta \neq \emptyset\) and doesn’t contain only 1 and \(\bot\), or \(\sigma \neq \emptyset\). Then it must be the case that \(P\) has some free name \(x : A \Delta \Delta\) where \(\sigma(x) = s[p][q]\). In this case, \(\rho\) and \(q\) are not implemented by the same thread and \(r\) and \(t\) are implemented by the same thread. It is easy to see we cannot compose such processes. The same reasoning applies as we increase \(n\) and \(m\). If \(t\), \(v\), or \(v\) are implemented by the same process, we must have processes sharing two distinct channels, which we cannot. If \(t\) and \(v\) are the same process, then the implementation of \(s\) must share two channels with this process, which is also a contradiction. Finally, if \(p\) and \(s\) are the same process, we are in the same situation as that described in the case in which \(n = 0\) and \(m = 0\).

**Lemma D.2.** Let \(P \llp_{\rho} \Delta ; \Delta ; \Gamma ; G\) with \(\Delta = \emptyset\) or containing only 1 or \(\bot\). We have that \(\text{co}(\Delta)\).

**Proof.** Assume to the contrary that \(\Gamma\) is not coherent. Then there exists \(s[p] : T_1 \sqsubseteq \Gamma\) such that \(T_1 \mid \bot \neq T' \mid p\). For this to be the case, let \(\bot \mid q\) contains an action unmatched by \(T' \mid p\) or vice-versa. Assume that \(T' \mid q\) contains an unmatched action. Since \(s[p] : T_1 \sqsubseteq \Gamma\) we know that there exists \(x : s[p][q] \in P = s[p][r] - \) this is necessarily the case due to the construction of the \(\llp\) judgment, insofar as if both \(s[p] : T'\) and \(s[p] : T_1 \sqsubseteq \Gamma\) with \(q \in \text{reset}(T)\) then we cannot introduce \(s[p]\) in the context unless with an instance of the composition rule for the corresponding channel \(x\) - where some sub-process of \(P\) uses \(x\):A for some \(A\) and some other sub-process of \(P\) uses \(x\):A'. The duality of \(x:\text{A'}\) and \(x:\text{A}\) contradicts the existence of an unmatched action in \(T\) \(q\). The argument for an unmatched action in \(T' \mid p\) is identical.

The other possibility is for \(s[p] : T_1 \sqsubseteq \Gamma\) such that \(q \in \text{roles}(T)\) and \(s[q] : T' \notin \Gamma\). If this is the case, then we must have \(x\) such that \(\sigma(x) = s[p][q]\), with \(x\):A in \(\Delta\) and \(A' \neq 1\), which is a contradiction.
Theorem D.2. Let $P \vdash_\sigma \Delta; \Gamma; G$. co($\Gamma$) iff $\Delta = \emptyset$ or $\Delta$ contains only 1 or $\bot$ and $\sigma = \emptyset$.

Proof. By Lemmas [D.1] and [D.2] ⊓⊔

Theorem D.3. Let $P \vdash_\sigma \Delta; \Gamma; G$ and $\Delta = \emptyset$ or $\Delta$ containing only 1 or $\bot$. Then we have: (1) $P \rightarrow ^* 0$; (2) fuse($G$) at each session is well-formed and deadlock-free; and (3) the interconnection network graph for fuse($G$) is acyclic.

Proof. (1) follows from [7, 27]. (2) follows from Prop. D.2. (3) follows from Theorem [D.1] ⊓⊔