Interconnectability of Session-based Logical Processes

BERNARDO TONINHO, NOVA-LINCS, Universidade Nova de Lisboa and Imperial College London, Portugal
NOBUKO YOSHIDA, Imperial College London, United Kingdom

In multiparty session types, interconnection networks identify which roles in a session engage in communication (i.e. two roles are connected if they exchange a message). In session-based interpretations of linear logic the analogue notion corresponds to determining which processes are composed, or cut, using compatible channels typed by linear propositions. In this work we show that well-formed interactions represented in a session-based interpretation of classical linear logic (CLL) form strictly less expressive interconnection networks than those of a multiparty session calculus. To achieve this result 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 threads. We then show that CLL composition induces PMC global types without circular interconnections between three (or more) participants. PMC is then used to define a new CLL composition rule which can form circular interconnections but preserves the deadlock-freedom of CLL.

CCS Concepts: • **Theory of computation** → **Distributed computing models**; **Process calculi**; *Linear logic*; • **Software and its engineering** → *Message passing*; *Concurrent programming languages*; *Concurrent programming structures*;

Additional Key Words and Phrases: Session Types, Classical Linear Logic, Multiparty Sessions, Synthesis

ACM Reference Format:
Bernardo Toninho and Nobuko Yoshida. 2018. Interconnectability of Session-based Logical Processes. *ACM Trans. Program. Lang. Syst.* 1, 1 (November 2018), 42 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

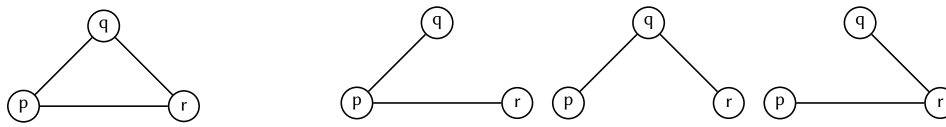
1 INTRODUCTION

The discovery of linear logic [28] and the early studies of its connections with concurrent processes [1, 2, 4] can be seen as the origin of a Curry-Howard correspondence for linear logic with typed interactive behaviours, which have led to the developments connecting linear logic and (binary) session types [8]. The understanding of linear logic propositions as session types [29], proofs as concurrent processes and proof simplification as communication not only has produced new logically-motivated techniques for reasoning about concurrent processes [51], but is also usable to articulate idioms of interaction with strong communication safety guarantees such as *protocol fidelity* and *deadlock freedom* [58]. The logical foundation of session types has also sparked a renewed interest on the theory and practice of session types [11, 14, 35, 39, 40].

The authors would like to thank the anonymous reviewers, Alceste Scalas and Julien Lange for their extensive comments and suggestions. This work is supported by NOVA LINCS (UID/CEC/04516/2013), EPSRC EP/K034413/1, EP/K011715/1, EP/L00058X/1, EP/N027833/1 and EP/N028201/1.

Authors' addresses: Bernardo Toninho, NOVA-LINCS, Universidade Nova de Lisboa and Imperial College London, Department of Computing, Portugal, btoninho@fct.unl.pt; Nobuko Yoshida, Imperial College London, Department of Computing, United Kingdom, n.yoshida@imperial.ac.uk.

0164-0925/2018/11-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn



(a) Multiparty (b) CLL

Fig. 1. Interconnection Networks of (1)

The term "interconnection networks" in this article originates from [2], denoting the connections between parallel threads (processes) by means of *linear* ports or channels. In standard 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 created 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 interconnections of processes: 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 this property that deadlock-freedom arises in a simple typing discipline, at the cost of disallowing more interesting interconnection networks.

This article provides a fresh look at session-based logical processes, based on concepts originating in *multiparty session types*. Motivated by an industry need [55] to specify protocols with more than two interconnected, interacting parties, the multiparty session types framework [31] develops a methodology where types implicitly describe connections between *many* communicating processes. The key 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 its interactions with all others parties) and check that each process adheres to its local type. Once all processes are typechecked, their composition can interact without deadlock, following the given global type. Recent work develops the connections of multiparty session types and communicating automata [6, 22, 37, 38], denotational semantics [21], Petri Nets [24], applications to, e.g. secure information flow analysis [10, 16], dynamic monitoring [5] and reversible computing [15]. Multiparty session types have also been integrated into mainstream programming languages such as MPI [41, 47], Java [32, 33, 42, 56], Python [20, 43, 46], C [49], Go [48], Erlang [25, 45, 57], Scala [54] and F# [44].

In multiparty sessions, interconnection networks identify which roles in a session engage in direct communication. Participant p is connected to another participant q iff p may exchange a message with q (or vice-versa). Consider the following 3-party interaction specified as a global type G:

$$G = \mathsf{p} \to \mathsf{q}{:}(\mathsf{nat}).\mathsf{p} \to \mathsf{r}{:}(\mathsf{bool}).\mathsf{r} \to \mathsf{q}{:}(\mathsf{str}).\mathsf{end} \tag{1}$$

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 network of Fig. 1a is *not* realisable in linear logic-based session frameworks, while those of Fig. 1b are.

We posit three processes with the behaviour ascribed by G, each implementing one role in the multiparty session:

$$P \vdash \mathsf{pq}{:}A, \mathsf{pr}{:}B \qquad Q \vdash \mathsf{pq}{:}A^{\perp}, \mathsf{qr}{:}C \qquad R \vdash \mathsf{pr}{:}B^{\perp}, \mathsf{qr}{:}C^{\perp}$$

where P is the implementation of p with channel 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, identified by the use of $A^{\perp}$) and qr for communication with r, and so on. While each process is individually typable, no 3-way composition is typable: composition in logic-based systems requires that the two processes share a single common channel name, which is then hidden under restriction. When we compose P and Q (hiding channel pq) we obtain a process that shares names pr and qr with R and so cannot be composed with it. We note that such an issue arises regardless of the order in which we choose to compose the processes. In essence, multiparty session types lead to richer connection topologies (e.g. circular connections) than those resulting from 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 by developing a framework based on the theory of multiparty session types (MP) that enables us to reason about connection topologies induced by the session-based interpretation of Classical Linear Logic (CLL).

Our framework is based on the observation that, since multiparty sessions subsume the binary sessions that are primitive in logical formulations of session types, it is possible to interpret typable processes in CLL as MP processes via a *structure-* and *typability-preserving* translation that maps CLL channels to MP channels that are indexed by a role and a destination (i.e. a consistent assignment of action prefixes in CLL to action prefixes in MP, identifying threads or cut-free CLL processes as individual MP session participants). Indeed, our mapping turns out to be canonical up-to bijective renaming.

In order to reason about the induced connection topologies, we build on the *synthesis* approaches to multiparty sessions [36, 37] which invert the projection-based proposals: instead of starting from a global type and then producing the certifiably deadlock-free local communication specifications, the works based on synthesis take a collection of local specifications (i.e. types) and study the conditions, dubbed *multiparty compatibility*, under which the local views form a deadlock-free global interaction. Our work extends these approaches by introducing a *compositional*, or partial, notion of multiparty compatibility (PMC) which allows us to synthesise a global type that represents the interactions obtained by iterated composition of CLL processes, providing the necessary tools to precisely study CLL connection topologies.

As argued above, we establish that process composition in CLL induces PMC global types without circular interconnections between three or more session participants (thus excluding the network of Fig. 1a). This result extends to other linear logic-based calculi (e.g. ILL [8]) since the fundamental structures induced by composition are the same as that of Fig. 1b, making precise the observation that well-formed interactions in linear logic-based calculi form strictly less expressive interconnections between participants than those of MP.

At a logical level, our observation is justified by the fact that allowing richer links or interconnections between proofs (i.e. processes) generally results in a failure of the cut elimination property, which reflects on the resulting processes as a failure of (global) progress. However, it is not the case that all such interconnection topologies result in deadlocked communication. Thus, given that PMC establishes sufficient conditions to ensure deadlock-freedom, even in the presence of circular interconnection topologies, we consider an extension to the CLL calculus in the form of a composition rule that can result in circular interconnection topologies, but that by being restricted to those of PMC global types, ensures deadlock-freedom (and, consequently, cut elimination) even in the presence of such richer links between proof objects.

Crucially, in contrast with other works on multiparty sessions and logic [11, 14], our PMC-based extension does not require modifications to the syntax of propositions or CLL processes. Also, previous work [11] gives an encoding (introduced earlier [7]) of their multiparty calculus into (binary) classical logic by using an *additional* orchestrator that centralises control of all interactions. Our canonical structure-preserving mapping requires no such processes and admits *more* typed representatives of given interconnection networks than existing works [11, 14] (see Example 6.9 and Example 4.13) even *without* the extended composition rule.

We note that as a consequence of our translation, we may use CLL to guarantee deadlock-freedom and termination of a class of MP processes with interleaved sessions, which is *not* normally guaranteed by MP typing systems [30] where only deadlock-freedom on a single session is ensured.

Contributions and outline:

- We introduce a structure- and typability-preserving translation of typed interactions in a session-based interpretation of CLL, restricted to processes without replication and higher-order channel passing, showing that the translation is *unique* insofar as there exists no other typability preserving encoding (up to bijective renaming) which maps an individual thread to a single participant (§ 3);
- We develop a compositional synthesis property, partial multiparty compatibility (PMC) (§ 4), which we use to show that the interconnectability of CLL is strictly less expressive than that of a single multiparty session in MP (§ 5);
- We systematically extend our results to the more intricate settings of higher-order channel passing (§ 6) and replication (§ 7), showing that neither feature enriches the interconnectability of CLL;
- We use PMC to develop an extension of CLL process composition dubbed multicut (§ 8) which enables richer interconnection topologies while preserving deadlock-freedom without modifying the types or syntax of CLL. We also show that our extended calculus is able to type a range of known examples from MP.

Our work does not assume a deep familiarity with the session-based interpretations of linear logic, multiparty session types or multiparty compatibility, providing introductions to the session calculi in § 2 and to global types and multiparty compatibility in § 4. An extended discussion of related work is given in § 9. The appendix lists additional proofs and definitions.

2 PROCESSES, TYPES AND TYPING SYSTEMS

This section introduces the two calculi used in our work: the binary session calculus CLL typed using the session type interpretation of classical linear logic [9, 60]; and the multiparty session calculus MP [18, 31]. In both settings, the notion of a session consists of a (predetermined) sequence of interactions on a given communication channel.

2.1 Classical Linear Logic (CLL) as Binary Session Types

We give a brief summary of the interpretation of classical linear logic as sessions, consisting of a variant of that of Wadler's CP [60], introduced in [9], using two context regions and without explicit contraction or weakening rules, which are admissible judgmental principles.

Syntax. The syntax of CLL processes $(P, Q, \ldots)$ is given below. Channels are ranged over by x, y, z, u, v, w, where we typically use x, y, z for *linear* channels and u, v, w for *shared* or replicated channels.

$$\begin{array}{rclll} P, Q & ::= & \overline{x}\langle y\rangle.P & \mid\; x(y).P & \text{Send and receive} \\ & \mid & x.l; P & \mid\; x.\mathsf{case}\{l_i : P\}_{i\in I} & \text{Selection and branching} \\ & \mid & \mathbf{0} & \mid\; (P \mid Q) & \text{Inaction and parallel} \\ & \mid & (\boldsymbol{\nu} x)P & \mid\; !u(y).P & \text{Hiding and replication} \end{array}$$

We consider a synchronous calculus with *fresh* (or bound) channel input and output – i.e., all sent channels are fresh by construction as in all works on logical session types (e.g. [8, 9, 40, 60]), following the internal mobility π-calculus [53]. The calculus also includes branching, selection and replication constructs, with the latter allowing us to represent servers as replicated input-guarded processes, where the corresponding matching output processes act as their clients. We write $fn(P)/bn(P)$ for the free/bound channels of P: in $\overline{x}\langle y\rangle.P$ and $(\boldsymbol{\nu} y)P$, y is a binding occurrence. We write $bv(P)$ for the bound variables of P, noting that in $x(y).P$ and $!x(y).P$, y is bound in P. We often omit $\mathbf{0}$.

Below we define the structural congruence for CLL, which is used in the typing system and the reduction semantics.

Definition 2.1 (Structural Congruence for CLL*).* Structural congruence of CLL processes, written $P \equiv Q$, is the least congruence defined by the following rules:

$$P \equiv_\alpha Q \Rightarrow P \equiv Q \qquad P \mid \mathbf{0} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R)$$
$$(\boldsymbol{\nu} x)(P \mid Q) \equiv (\boldsymbol{\nu} x)P \mid Q \quad x \notin fn(Q) \qquad (\boldsymbol{\nu} x)(!x(y).P) \equiv \mathbf{0}$$

Reduction. The reduction semantics for CLL, written $P \to Q$ and defined up to structural congruence $\equiv$, is given below:

$$\begin{array}{rcl} \overline{x}\langle y\rangle.P \mid x(y).Q & \to & (\boldsymbol{\nu} y)(P \mid Q) \\ x.l_j; P \mid x.\mathsf{case}\{l_i{:}Q_i\}_{i\in I} & \to & P \mid Q_j \quad (j \in I) \\ \overline{x}\langle y\rangle.P \mid !x(y).Q & \to & (\boldsymbol{\nu} y)(P \mid Q) \mid !x(y).Q \\ P \to P' & \Rightarrow & P \mid Q \to P' \mid Q \\ P \to P' & \Rightarrow & (\boldsymbol{\nu} x)P \to (\boldsymbol{\nu} x)P' \\ P \equiv P' \wedge P' \to Q' \wedge Q' \equiv Q & \Rightarrow & P \to Q \end{array}$$

Definition 2.2 (Live Process). A process P is live, written $live(P)$ iff $P \equiv (\boldsymbol{\nu}\tilde{x})(\pi.Q \mid R)$ or $P \equiv (\boldsymbol{\nu}\tilde{x})(\pi; Q \mid R)$ for some R, sequences of names $\tilde{x}$ and a *non-replicated* guarded process $\pi.Q$ or $\pi; Q$, where π is any non-replicated process prefix.

Note how the definition of live process excludes a process of the form $!u(y).P$, which corresponds to a replicated server that has no remaining users.

Types. The syntax of (logical) binary session types A, B is:

$$A, B \quad ::= \quad A \otimes B \mid A \mathbin{⅋} B \mid \mathbf{1} \mid \bot \mid \oplus\{l_i : A_i\}_{i\in I} \mid \&\{l_i : A_i\}_{i\in I} \mid ?A \mid !A$$

Following [9, 60], $\otimes$ corresponds to output of a session of type A followed by behaviour B; $⅋$ to input of A followed by B; $\oplus$ and $\&$ to selection and branching; $!A$ to replicated channels of type A (i.e. persistent servers) and $?A$ to clients of such servers. The *dual* of A, written $A^\perp$, is defined as (we omit the involutive cases):

$$\mathbf{1}^\perp \triangleq \bot,\ (A \otimes B)^\perp \triangleq A^\perp \mathbin{⅋} B^\perp,\ (\oplus\{l_i{:}A_i\}_{i\in I})^\perp \triangleq \&\{l_i{:}A_i^\perp\}_{i\in I},\ (!A)^\perp \triangleq ?A^\perp.$$

Typing system. We define the typing system CLL in Fig. 2, assigning the usage of channels in P processes to types A, B. The typing judgement is written $P \vdash_{\mathsf{CL}} \Xi; \Delta$, defined *up to structural congruence* $\equiv$ (i.e. we implicitly have that if $P \vdash_{\mathsf{CL}} \Xi; \Delta$ and $P \equiv Q$ then $Q \vdash_{\mathsf{CL}} \Xi; \Delta$), where Δ is a set of hypotheses of the form $x{:}A$ (not subject to weakening or contraction), where x stands for a free session channel in P and A is a binary session type; and Ξ is a set of hypotheses of the form $u{:}A$, subject to weakening and contraction, standing for the channels used in P in an unrestricted (or shared) manner. The typing judgement states that process P uses channels according to the session discipline ascribed by Δ and Ξ. We assume all channel names in Δ and Ξ are distinct. We write $\cdot$ for the empty typing environment and Δ, Δ' for the union of Δ and Δ', only defined when channels in Δ and Δ' are distinct.

Rule ($\otimes$) accounts for the session output behaviour, typing a channel x with $A \otimes B$ if the process outputs along x a name y that is used in P_1 with behaviour A and, disjointly, P_2 uses x according to B (this strict separation is crucial for deadlock-freedom); dually, rule ($\mathbin{⅋}$) types a channel x with $A \mathbin{⅋} B$ if the process performs an input on x of a session channel y such that y is used in the continuation as A, and x as B; rule ($\mathbf{1}$) types the inactive process with an arbitrary session channel assigned type $\mathbf{1}$; rule ($\perp$) types the dual behaviour, which just discards the no longer used name; rule ($\oplus$) types channel x with $\oplus\{l_i{:}A_i\}_{i \in I}$ by having the process emit a label l_j with $j \in I$, and then using the channel x according to the type A_j in the corresponding branch; dually, rule ($\&$) types processes that wait for a choice on channel x, with type $\&\{l_i{:}A_i\}_{i \in I}$, if the process can account for all of the possible choice labels and corresponding behaviours in the type. Thus, the case construct must contain one process P_i using x according to behaviour A_i for each label in the type. Note the additive nature of the rule, where the context Δ is the same in all premises. This enforces that all possible alternative behaviours make use of the *same* session behaviours.

Rule (cut) composes in parallel two processes P and Q, that use channel x with dual types A and $A^\perp$, by hiding x in the composed process in the conclusion of the rule (since no other process may use x). We note that Δ and Δ' are disjoint, so the only common channel between P and Q is x.

The remaining rules define the typing for the replication constructs. Rule (!) types a replicated input channel u as $!A$ if the continuation P uses the input as A without using any other linear channels, which ensures that the replicas of P do not invalidate the linear typing discipline. Rule (?) moves a session channel of type $?A$ to the appropriate shared context Ξ as A, renaming it to u. Rule (copy) types a usage of a shared channel u by sending a fresh channel x along u, which is then used (linearly) as A in the continuation P. Rule ($\text{cut}^!$) allows for composition of shared sessions, provided the processes use no linear channels. Such a rule is needed to ensure that cut elimination holds structurally (i.e. a linear cut between a session channel of type $!A$ and its dual $?A^\perp$ reduces to a $\text{cut}^!$, which will eventually reduce back to a cut). We note that at the level of typable processes, $\text{cut}^!$ can be represented as a cut between rules ! and ?.

Proposition 2.3 (Deadlock-freedom in CLL [9, 60]). *Suppose $P \vdash_{\mathsf{CL}} \cdot; \Delta$, with $live(P)$ where Δ is either empty or only contains $\mathbf{1}$ or $\perp$. We have that $P \to P'$.*

2.2 Multiparty Session (MP) Calculus

Syntax. We introduce the MP calculus of multiparty sessions, where processes P, Q use channels indexed by roles of the multiparty sessions in which they are used. The syntax of processes and channels is given below:

$$
\begin{array}{lcll}
P, Q & ::= & c[\mathtt{p}]\langle c' \rangle; P \mid c[\mathtt{p}](x); P & \text{Send and receive} \\
& \mid & c[\mathtt{p}] \oplus l; P \ \mid c[\mathtt{p}] \ \& \ \{l_i{:}P_i\}_{i \in I} & \text{Selection and branching} \\
& \mid & \mathbf{0} \mid (P \mid Q) \mid (\boldsymbol{\nu} s)P & \text{Inaction, parallel, hiding} \\
c & ::= & x \mid s[\mathtt{p}] & \text{variable, role-indexed channel}
\end{array}
$$

$$(\otimes)\frac{P_1 \vdash_{\mathsf{CL}} \Xi; \Delta, y{:}A \quad P_2 \vdash_{\mathsf{CL}} \Xi; \Delta', x{:}B}{\overline{x}\langle y\rangle.(P_1 \mid P_2) \vdash_{\mathsf{CL}} \Xi; \Delta, \Delta', x{:}A \otimes B} \qquad (\invamp)\frac{P \vdash_{\mathsf{CL}} \Xi; \Delta, y{:}A, x{:}B}{x(y).P \vdash_{\mathsf{CL}} \Xi; \Delta, x{:}A \invamp B}$$

$$(\mathbf{1})\frac{}{\mathbf{0} \vdash_{\mathsf{CL}} \Xi; x{:}\mathbf{1}} \qquad (\bot)\frac{P \vdash_{\mathsf{CL}} \Xi; \Delta}{P \vdash_{\mathsf{CL}} \Xi; \Delta, x{:}\bot}$$

$$(\oplus)\frac{P \vdash_{\mathsf{CL}} \Xi; \Delta, x{:}A_j \quad j \in I}{x.l_j; P \vdash_{\mathsf{CL}} \Xi; \Delta, x{:} \oplus \{l_i : A_i\}_{i\in I}} \qquad (\&)\frac{P_1 \vdash_{\mathsf{CL}} \Xi; \Delta, x{:}A_1 \quad \dots \quad P_n \vdash_{\mathsf{CL}} \Xi; \Delta, x{:}A_n}{x.\mathsf{case}\{l_i : P_i\}_{i\in I} \vdash_{\mathsf{CL}} \Xi; \Delta, x{:}\ \&\ \{l_i : A_i\}_{i\in I}}$$

$$(\mathsf{cut})\frac{P \vdash_{\mathsf{CL}} \Xi; \Delta, x{:}A \quad Q \vdash_{\mathsf{CL}} \Xi; \Delta', x{:}A^\perp}{(\boldsymbol{\nu} x)(P \mid Q) \vdash_{\mathsf{CL}} \Xi; \Delta, \Delta'}$$

$$(!)\frac{P \vdash_{\mathsf{CL}} \Xi; y{:}A}{!u(y).P \vdash_{\mathsf{CL}} \Xi; u{:}!A} \qquad (?)\frac{P \vdash_{\mathsf{CL}} \Xi, u{:}A; \Delta}{P\{x/u\} \vdash_{\mathsf{CL}} \Xi; \Delta, x{:}?A}$$

$$(\mathsf{copy})\frac{P \vdash_{\mathsf{CL}} \Xi, u{:}A; \Delta, x{:}A}{\overline{u}\langle x\rangle.P \vdash_{\mathsf{CL}} \Xi, u{:}A; \Delta} \qquad (\mathsf{cut}^!)\frac{P \vdash_{\mathsf{CL}} \Xi; x{:}A \quad Q \vdash_{\mathsf{CL}} \Xi, u{:}A^\perp; \Delta}{(\boldsymbol{\nu} u)(!u(x).P \mid Q) \vdash_{\mathsf{CL}} \Xi; \Delta}$$

Fig. 2. CLL Typing Rules

Role names are identified by p, q, r; channels are ranged over by s, t; c denotes channels with role $s[\mathrm{p}]$ or variables x.

Local types. Role indexed channels $s[\mathrm{p}]$ in MP are assigned *local types*, ranged over by S, T, denoting the behaviour of each role per channel. Local types are defined as follows:

$$S, T ::= \mathrm{p}\uparrow(T); S \mid \mathrm{p}\downarrow(T); S \mid \oplus\mathrm{p}\{l_i{:}T_i\}_{i\in I} \mid \&\mathrm{p}\{l_i{:}T_i\}_{i\in I} \mid \mathsf{end}$$

The local types $\mathrm{p}\uparrow(T); S$ and $\mathrm{p}\downarrow(T); S$, which type the send and receive constructs above, denote output to and input from role p of a session channel of type T, followed by behaviour S, respectively. Types $\oplus\mathrm{p}\{l_i{:}T_i\}_{i\in I}$ and $\&\mathrm{p}\{l_i{:}T_i\}_{i\in I}$, which type the selection and branching constructs, denote the emission (resp. reception) of a label l_i to (resp. from) role p, followed by behaviour T_i. Type end denotes no further behaviour. We define the set of roles of local type T, denoted by $\mathsf{roles}(T)$, as the set of all roles occurring in type T.

Partial projection and coherence. To define the typing system for MP, we introduce partial projection and coherence. Partial projection takes a local type (which specifies all interactions for a given role) and a role to produce the binary session type [29] that corresponds to the interactions between the role whose behaviour is denoted by the local type and the given role, from the point of view of the former (e.g. if p is the role behaving according to T_p, the projection of T_p for q produces a binary session type describing the interactions between p and q from the perspective of p). Binary session types S, T are given by (by abuse of notation we re-use the same symbols S, T as for local types):

$$\uparrow(T); S \qquad \downarrow(T); S \qquad \oplus\ \{l_i{:}T_i\}_{i\in I} \qquad \&\ \{l_i{:}T_i\}_{i\in I} \qquad \mathsf{end}$$

and their notion of duality $\overline{T}$ is given by: $\overline{\uparrow(T); S} \triangleq\ \downarrow(T); \overline{S}$, $\overline{\oplus\{l_i{:}T_i\}_{i\in I}} \triangleq \&\{l_i{:}\overline{T_i}\}_{i\in I}$, $\overline{\mathsf{end}} \triangleq \mathsf{end}$; and the involutive rules for $\downarrow$ and $\&$.

Definition 2.4 (Partial Projection). Given a local type T, we define its partial projection onto a participant p, written $T{\upharpoonright}\mathsf{p}$, by induction on the structure of T by:

$$
\begin{array}{llll}
(\mathsf{r}{\uparrow}(S);T) \upharpoonright \mathsf{p} = \begin{cases} \uparrow(S);(T \upharpoonright \mathsf{p}) & \text{if } \mathsf{p}=\mathsf{r} \\ T \upharpoonright \mathsf{p} & \text{otherwise} \end{cases} & (\oplus \mathsf{r}\{l_i{:}T_i\}_{i\in I}) \upharpoonright \mathsf{p} = \begin{cases} \oplus\{l_i{:}(T_i \upharpoonright \mathsf{p})\}_{i\in I} & \text{if } \mathsf{p}=\mathsf{r} \\ \sqcup_{i\in I}(T_i \upharpoonright \mathsf{p}) & \text{otherwise} \end{cases} \\
(\mathsf{r}{\downarrow}(S);T) \upharpoonright \mathsf{p} = \begin{cases} \downarrow(S);(T \upharpoonright \mathsf{p}) & \text{if } \mathsf{p}=\mathsf{r} \\ T \upharpoonright \mathsf{p} & \text{otherwise} \end{cases} & (\& \mathsf{r}\{l_i{:}T_i\}_{i\in I}) \upharpoonright \mathsf{p} = \begin{cases} \&\{l_i{:}(T_i \upharpoonright \mathsf{p})\}_{i\in I} & \text{if } \mathsf{p}=\mathsf{r} \\ \sqcup_{i\in I}(T_i \upharpoonright \mathsf{p}) & \text{otherwise} \end{cases} \\
\quad \mathsf{end} \upharpoonright \mathsf{p} = \ \mathsf{end} &
\end{array}
$$

where the *merge* $T \sqcup T'$ of T and T' is defined by $T \sqcup T \triangleq T$; and with $T = \oplus\{\mathsf{l}_i : T_i\}_{i\in I}$ and $T' = \oplus\{\mathsf{l}'_j : T'_j\}_{j\in J}$,

$$T \sqcup T' \triangleq \oplus(\{\mathsf{l}_h : T_h\}_{h\in I\setminus J} \cup \{\mathsf{l}'_h : T'_h\}_{h\in J\setminus I} \cup \{\mathsf{l}_h : T_h \sqcup T'_h\}_{h\in I\cap J})$$

if $\mathsf{l}_h = \mathsf{l}'_h$ for each $h \in I \cap J$; and homomorphic for other types (i.e. $\mathcal{T}[T_1] \sqcup \mathcal{T}[T_2] = \mathcal{T}[T_1 \sqcup T_2]$ where $\mathcal{T}$ is a context of local types). $T \sqcup T'$ is undefined otherwise. Partial projection is undefined if merging is undefined.

Merging is needed for two purposes: (1) to check global types well-formedness (i.e. if merge is undefined then the global type is not well-formed); and (2) to allow for more typable protocols. Examples of merging can be found in § 4.1. Coherence ensures that the local types of interacting roles contain the necessary compatible actions (e.g. if the local type for p specifies an emission to q, the local type for q specifies a reception from p [18, 31]) and all the necessary roles are ascribed a type in the context. To define coherence, we introduce session subtyping. We note that the subtyping relation is inverted w.r.t. the "process-oriented" subtyping [26] because, for convenience, we adopt the "channel-oriented" ordering [17]; an analysis of the two subtyping relations is given in [27].

Definition 2.5 (Session Subtyping). We define the subtyping relation between binary session types, $T \leq S$, as the least relation given by the following rules:

$$
\frac{}{\mathsf{end} \leq \mathsf{end}} \qquad \frac{\forall i \in I \quad T_i \leq T'_i}{\oplus\{l_i : T_i\}_{i\in I} \leq \oplus\{l_i : T'_i\}_{i\in I\cup J}} \qquad \frac{\forall i \in I \quad T_i \leq T'_i}{\&\{l_i : T_i\}_{i\in I\cup J} \leq \&\{l_i : T'_i\}_{i\in I}}
$$

$$
\frac{T \leq T' \quad S \leq S'}{\uparrow(T);S \leq \uparrow(T');S'} \qquad \frac{T' \leq T \quad S \leq S'}{\downarrow(T);S \leq \downarrow(T');S'}
$$

Definition 2.6 (Coherence). Γ is coherent (denoted by $\mathsf{co}(\Gamma)$) iff $s[\mathsf{p}]{:}T_1 \in \Gamma$ and $s[\mathsf{q}]{:}T_2 \in \Gamma$ with $\mathsf{p} \neq \mathsf{q}$ imply that $T_1 \upharpoonright \mathsf{q} \leq \overline{T_2 \upharpoonright \mathsf{p}}$; and for all $s[\mathsf{p}]{:}T \in \Gamma$ and $\mathsf{q} \in \mathsf{roles}(T)$, $s[\mathsf{q}]{:}T' \in \Gamma$, for some T'.

Typing rules. We define the typing system MP in Fig. 3, assigning the usage of role-indexed channels to local types. The judgement $P \vdash_{\mathsf{MP}} \Gamma$, where Γ is a set of hypotheses of the form $c{:}T$, denotes that P uses its channels according to Γ. We assume the same notations and conditions for Γ and Γ, Γ' as for CLL, where $\cdot$ denotes the empty context and Γ, Γ' denotes the disjoint union of Γ and Γ', defined only when their domains are disjoint.

Rule (end) types the inactive process in a session context containing only terminated sessions (i.e. the context Γ is a collection of assumptions of the form $c_i{:}\mathsf{end}$). Rule (send) types the emission of a channel endpoint of type T to role q, assigning c the local type $\mathsf{q}{\uparrow}(T);S$, provided the continuation P uses c according to type S. Dually, rule (recv) types the reception of a value of type T, bound to x in the continuation P, sent by q, with type $\mathsf{q}{\downarrow}(T);S$, provided P uses c according to S. Rules (sel) and (bra) are the MP counterparts of rules ($\oplus$) and ($\&$) from CLL (Fig. 2), respectively, with the former typing the emission of a label to q and the latter typing the reception of a label from q. Rule

$$\text{(end)}\frac{\Gamma \text{ end only}}{\mathbf{0} \vdash_{\mathsf{MP}} \Gamma} \quad \text{(send)}\frac{P \vdash_{\mathsf{MP}} \Gamma, c{:}S}{c[\mathsf{q}]\langle c' \rangle; P \vdash_{\mathsf{MP}} \Gamma, c{:}\mathsf{q}\uparrow(T); S, c'{:}T} \quad \text{(recv)}\frac{P \vdash_{\mathsf{MP}} \Gamma, c{:}S, x{:}T}{c[\mathsf{q}](x); P \vdash_{\mathsf{MP}} \Gamma, c{:}\mathsf{q}\downarrow(T); S}$$

$$\text{(sel)}\frac{P \vdash_{\mathsf{MP}} \Gamma, c{:}T_j \quad j \in I}{c[\mathsf{q}] \oplus l_j; P \vdash_{\mathsf{MP}} \Gamma, c{:} \oplus \mathsf{q}\{l_i : T_i\}_{i \in I}} \quad \text{(bra)}\frac{P_1 \vdash_{\mathsf{MP}} \Gamma, c{:}T_1 \quad \dots \quad P_n \vdash_{\mathsf{MP}} \Gamma, c{:}T_n}{c[\mathsf{q}] \,\&\, \{l_i{:}P_i\}_{i \in I} \vdash_{\mathsf{MP}} \Gamma, c{:} \,\&\, \mathsf{q}\{l_i : T_i\}_{i \in I}}$$

$$\text{(comp)}\frac{P \vdash_{\mathsf{MP}} \Gamma \quad Q \vdash_{\mathsf{MP}} \Gamma'}{P \mid Q \vdash_{\mathsf{MP}} \Gamma, \Gamma'} \quad \text{(close)}\frac{P \vdash_{\mathsf{MP}} \Gamma, s[\mathsf{p}_1] : T_1, .., s[\mathsf{p}_n] : T_n \quad \mathsf{co}(s[\mathsf{p}_1]{:}T_1, .., s[\mathsf{p}_n]{:}T_n)}{(\boldsymbol{\nu} s)P \vdash_{\mathsf{MP}} \Gamma}$$

Fig. 3. MP Typing Rules

(comp) types parallel composition of processes with disjoint session contexts Γ and Γ'. Rule (close) types a *coherent* multiparty session s by hiding the session channel, provided that process P uses $s[\mathsf{p}_1]{:}T_1, \dots, s[\mathsf{p}_n]{:}T_n$ and the corresponding role indices and local types form a coherent typing context.

Reduction. The reduction semantics for MP processes is given below (omitting closure under structural congruence). They are fundamentally identical to the reduction rules of CLL, but require not just the session channel to match but also the role assignment to be consistent:

$$\begin{array}{rcl} s[\mathsf{p}][\mathsf{q}]\langle s'[\mathsf{p}'] \rangle; P \mid s[\mathsf{q}][\mathsf{p}](x); Q & \to & P \mid Q\{s'[\mathsf{p}']/x\} \\ s[\mathsf{p}][\mathsf{q}] \oplus l_j; P \mid s[\mathsf{q}][\mathsf{p}] \,\&\, \{l_i{:}Q_i\}_{i \in I} & \to & P \mid Q_j \quad (j \in I) \\ P \to P' & \Rightarrow & P \mid Q \to P' \mid Q \\ P \to P' & \Rightarrow & (\boldsymbol{\nu} s)P \to P' \end{array}$$

We highlight that, in contrast to CLL, the typing system MP alone does *not* ensure deadlock-freedom, where deadlock-freedom means that all communication actions always eventually fire for processes typed in an empty context. We assume basic value passing, noting that value passing can be encoded with terminated sessions and that henceforth it will be used freely in the rest of the paper.

Proposition 2.7 (Deadlock in MP). *There exists a deadlocked process P that is typable in* MP, *i.e.* $P \vdash_{\mathsf{MP}} \emptyset$ *does* not *imply that P is deadlock-free.*

Proof. Take $P = s[\mathsf{p}][\mathsf{r}](x); s[\mathsf{p}][\mathsf{q}]\langle 7 \rangle$, $Q = s[\mathsf{q}][\mathsf{p}](x); s[\mathsf{q}][\mathsf{r}]\langle \mathtt{tt} \rangle$ and $R = s[\mathsf{r}][\mathsf{q}](x); s[\mathsf{r}][\mathsf{p}]\langle \text{"}a\text{"} \rangle$. $(\boldsymbol{\nu} s)(P \mid Q \mid R) \vdash_{\mathsf{MP}} \emptyset$, but $P \mid Q \mid R$ is deadlocked. □

3 RELATING THE CLL AND MP SYSTEMS

In this section we develop one of our main contributions: the connection between the CLL and MP systems. For presentation purposes, we first consider a restriction of CLL *without* name passing and replication, which are addressed in § 6 and § 7, respectively. In the following sections we tacitly make use of value passing, which can be included straightforwardly in the systems of § 2. To explicate our approach, consider the following CLL typable processes:

$$\begin{array}{l} P \triangleq x\langle 7 \rangle.y(z).x\langle \text{"}hello\text{"} \rangle.\mathbf{0} \vdash_{\mathsf{CL}} x{:}\mathsf{nat} \otimes \mathsf{str} \otimes \mathbf{1}, y{:}\mathsf{nat} \parr \bot \\ P' \triangleq y(z).x\langle 7 \rangle.x\langle \text{"}hello\text{"} \rangle.\mathbf{0} \vdash_{\mathsf{CL}} x{:}\mathsf{nat} \otimes \mathsf{str} \otimes \mathbf{1}, y{:}\mathsf{nat} \parr \bot \end{array}$$

Both P and P' are typable in the same context, however P first outputs on x, then inputs on y and then outputs on x again, whereas P' flips the order of the first two actions. By the nature of process composition in CLL, both processes can be safely composed with any typable $R_1 \vdash_{\mathsf{CL}} x{:}\mathsf{nat} \parr \mathsf{str} \parr \bot$ and $R_2 \vdash_{\mathsf{CL}} y{:}\mathsf{nat} \otimes \mathbf{1}$. We also observe that, since both P and P' are typable in the same context, CLL typing cannot capture *cross-channel sequential dependencies* (i.e. it cannot distinguish orderings of actions on different channels).

We now consider a mapping from CLL to MP. The following processes Q and Q' are hypothetical translations of P and P'. The notation $s[\mathsf{p}][\mathsf{q}]$ represents a channel in session s with role p and destination q:

$$Q \triangleq s[\mathsf{q}][\mathsf{p}]\langle 7\rangle; s[\mathsf{q}][\mathsf{r}](z); s[\mathsf{q}][\mathsf{p}]\langle\text{"}hello\text{"}\rangle \quad Q' \triangleq s[\mathsf{q}][\mathsf{r}](z); s[\mathsf{q}][\mathsf{p}]\langle 7\rangle; s[\mathsf{q}][\mathsf{p}]\langle\text{"}hello\text{"}\rangle$$

The processes P and Q above are similar, insofar as both send 7 to a destination (resp. x and role p), followed by an input (resp. on y and from r), followed by an output of "hello" to the initial destination. A similar argument can be made for P' and Q'. Despite P and P' having the same types, we have:

$$Q \vdash_{\mathsf{MP}} s[\mathsf{q}]{:}\mathsf{p}\uparrow(\mathsf{nat}); \mathsf{r}\downarrow(\mathsf{nat}); \mathsf{p}\uparrow(\mathsf{str}); \mathsf{end} \quad Q' \vdash_{\mathsf{MP}} s[\mathsf{q}]{:}\mathsf{r}\downarrow(\mathsf{nat}); \mathsf{p}\uparrow(\mathsf{nat}); \mathsf{p}\uparrow(\mathsf{str}); \mathsf{end}$$

By refining channels with role annotations, MP distinguishes orderings of actions on different session *sub-channels* (i.e. the communication links between the several role pairs). More precisely, by grouping the actions of role q along its two session sub-channels $s[\mathsf{q}][\mathsf{p}]$ and $s[\mathsf{q}][\mathsf{r}]$ at the type level, we can precisely track the ordering and causal dependencies on the message exchanges between q and p and those with r. Thus, our goal is to find a precise way to systematically map process P to process Q (and P' to Q'), and also generate the corresponding local typing in a typability preserving way.

To relate CLL with MP processes and preserve typability, we proceed as follows:

Mapping 1: $\boxed{P \vdash^{\sigma}_{\mathsf{CL}} \Delta}$ We define a mapping σ from *session channels* in CLL to *channels indexed with role and destination* in a single MP session, such that given a single-threaded process in CLL (i.e, a cut-free process), we map its channels to role and destination-annotated channels in MP forming a single multiparty session, capturing the cross-channel causal dependencies that are not codified at the level of CLL types.

Mapping 2: $\boxed{[\![P]\!]_{\sigma}}$ We generate local type T from a single thread P wrt. σ such that $P \vdash^{\sigma}_{\mathsf{CL}} \Delta$ so that we can translate P in CLL to $\sigma(P)$ typable in MP.

Mapping 3: $\boxed{P \Vdash^{\sigma}_{\rho} \Delta; \Gamma}$ We translate the cut (i.e. parallel composition) between two processes in CLL into MP generating a mapping from channels to *session* types Γ with renaming of free and bound names (σ and ρ). This automatically provides a type- and thread-preserving translation $\rho(\sigma(P))$ into MP, which is *unique* up to bijective renaming.

Mapping 1: Preservation of threads and typability. Definition 3.1 provides the mapping from session channels in CLL to those in MP. For now, we consider only CLL processes without replication (i.e. typed without uses of rules !, ?, copy and $\mathrm{cut}^!$ – and thus omit Ξ from the CLL typing judgment) and where $A \otimes B$ and $A ⅋ B$ are restricted to $\mathbf{1} \otimes B$ and $\perp ⅋ B$, respectively (i.e. no higher-order channel passing, where $\mathbf{1} \otimes B$ can be seen as sending an abstract value of ground type). Moreover, we assume that uses of rule ($\otimes$) are such that $P_1 \equiv \mathbf{0}$. We lift the restriction on higher-order channel passing in § 6.

Definition 3.1 (Channel to Role-Indexed Channel Mapping). Let $P \vdash_{\mathsf{CL}} \Delta$ such that the typing derivation does not use the cut rule. We define a channel to (role-)indexed channel mapping σ such that for all $x, y \in fn(P)$, if $x \neq y$, then $\sigma(x) = s[\mathsf{p}][\mathsf{q}]$ and $\sigma(y) = s[\mathsf{p}][\mathsf{q}']$, for some q, q' such that $\mathsf{q} \neq \mathsf{q}'$, and *unique* s and p (i.e. s and p are the same MP session channel and principal role across the entire mapping σ). We reject reflexive role assignments of the form $s[\mathsf{p}][\mathsf{p}]$.

We write $P \vdash^{\sigma}_{\mathsf{CL}} \Delta$ to denote such a mapping and $\mathsf{c}_{\sigma}(x)$, $\mathsf{p}_{\sigma}(x)$ and $\mathsf{d}_{\sigma}(x)$ to denote the channel, first (principal) and second (destination) roles in the image of x in σ.

$$\text{(thread)}\,\frac{P \vdash^{\sigma}_{\mathsf{CL}} \Delta}{P \Vdash^{\sigma}_{\emptyset} \Delta; s[\mathsf{p}_\sigma]{:}[\![P]\!]_\sigma} \qquad \text{(comp)}\,\frac{P \vdash^{\sigma}_{\mathsf{CL}} \Delta, x{:}A \quad Q \Vdash^{\sigma'}_{\rho} \Delta', x{:}A^{\perp}; \Gamma \quad (\star)}{(\boldsymbol{\nu} x)(P \mid Q) \Vdash^{(\sigma' \cup \sigma)\setminus\{x\}}_{\rho'} \Delta, \Delta'; \Gamma, s[\mathsf{p}_\sigma]{:}[\![P]\!]_\sigma}$$

$(\star)$ **(a)** bound channels: $\rho' = \rho \cup (x, s[\mathsf{p}_\sigma][\mathsf{d}_\sigma(x)])$
(b) role/destination match: $\mathsf{p}_\sigma(x) = \mathsf{d}_{\sigma'}(x) \wedge \mathsf{d}_\sigma(x) = \mathsf{p}_{\sigma'}(x)$
(c) unique destinations: $\forall z \in \Delta, y \in \Delta'.\mathsf{d}_\sigma(z) \neq \mathsf{d}_{\sigma'}(y) \wedge \mathsf{d}_\sigma(z), \mathsf{d}_{\sigma'}(y) \notin \rho$

Fig. 4. Parallel Composition Mapping

A mapping according to Definition 3.1 identifies a single-threaded process of CLL with a single role implementation in MP, such that all its channels are mapped to the same multiparty session channel s and same principal role p, but to different destination roles.

Convention 3.1. *In the remainder of this section and § 5, given $P \vdash^{\sigma}_{\mathsf{CL}} \Delta$, we assume $\forall x, y \in fn(P), \mathsf{c}_\sigma(x) = \mathsf{c}_\sigma(y) = s$ and $\mathsf{p}_\sigma(x) = \mathsf{p}_\sigma(y) = \mathsf{p}_\sigma$. This convention is allowed due to the session and principal role for all channels in a given mapping σ being constant. This assumption is lifted in § 6.*

Let $P \vdash^{\sigma}_{\mathsf{CL}} \Delta$. We write $\sigma(P)$ for the process obtained by renaming each free name x in P with $\sigma(x)$, where actions in P are mapped to their corresponding actions in MP:

$$\begin{array}{rclrcl}
\sigma(x(y).P) & \triangleq & s[\mathsf{p}_\sigma][\mathsf{d}_\sigma(x)](y); \sigma(P) & \sigma(\overline{x}\langle y\rangle.P) & \triangleq & s[\mathsf{p}_\sigma][\mathsf{d}_\sigma(x)]\langle y\rangle; \sigma(P) \\
\sigma(x.l_j; P) & \triangleq & s[\mathsf{p}_\sigma][\mathsf{d}_\sigma(x)] \oplus l_j; \sigma(P) & \sigma(x.\mathsf{case}\{l_i{:}Q_i\}_{i\in I}) & \triangleq & s[\mathsf{p}_\sigma][\mathsf{d}_\sigma(x)] \,\&\, \{l_i{:}\sigma(Q_i)\}_{i\in I}
\end{array}$$

Mapping 2: Generating local types. Having constructed a syntactic mapping from CLL to MP processes, we now present a way to generate the appropriate local typings for processes in the image of the translation.

Definition 3.2 (Local Type Generation). Let $P \vdash^{\sigma}_{\mathsf{CL}} \Delta$. We generate a local type T such that $\sigma(P) \vdash_{\mathsf{MP}} s[\mathsf{p}_\sigma]{:}T$ by induction on the structure of P, written $[\![P]\!]_\sigma$ (assume $\mathsf{d}_\sigma(x) = \mathsf{q}$ and $S = \mathsf{end}$, noting that value passing is encoded by the communication of sessions of type end):

$$\begin{array}{c}
[\![\mathbf{0}]\!]_\sigma \triangleq \mathsf{end} \qquad [\![\overline{x}\langle y\rangle.P]\!]_\sigma \triangleq \mathsf{q}\uparrow(S); [\![P]\!]_\sigma \qquad [\![x(y).P]\!]_\sigma \triangleq \mathsf{q}\downarrow(S); [\![P]\!]_\sigma \\
[\![x.l_j; P]\!]_\sigma \triangleq \oplus\mathsf{q}\{l_j{:}[\![P]\!]_\sigma\} \qquad [\![x.\mathsf{case}\{l_i{:}P_i\}_{i\in I}]\!]_\sigma \triangleq \&\mathsf{q}\{l_i{:}[\![P_i]\!]_\sigma\}_{i\in I}
\end{array}$$

Hence, given a cut-free $P \vdash_{\mathsf{CL}} \Delta$, we have an automatic way of generating a renaming σ such that $P \vdash^{\sigma}_{\mathsf{CL}} \Delta$ and $\sigma(P) \vdash_{\mathsf{MP}} s[\mathsf{p}_\sigma]{:}T$ with $T = [\![P]\!]_\sigma$.

Mapping 3: Parallel composition. Fig. 4 defines the judgement $P \Vdash^{\sigma}_{\rho} \Delta; \Gamma$ such that P is an n-ary composition of processes, Γ is an MP session typing context, Δ is a CLL linear context, σ (resp. ρ) is a mapping from free (resp. bound) names to indexed channels. Recall that s stands for the (unique) channel in the mappings σ and σ'. Rule (comp) defines the composition of a single-thread CLL process with an n-ary composition of CLL processes which can be mapped to MP typed processes. The rule ensures that the resulting process is well-formed in both CLL and MP: clause **(a – bound channel)** constructs the mapping ρ' for bound channels, as they are hidden by CLL composition; **(b – role/destination match)** ensures that σ and σ' map x to the same multiparty session channel, where the destination role in $\sigma(x)$ matches the principal role in $\sigma'(x)$, and vice-versa; **(c – unique destinations)** asserts that channels in Δ and Δ' cannot have the same destination role, ensuring uniqueness of common channels, and that free name assignments do not capture those of bound names.

We write $\rho(P)$ for the renaming of bound names in P generated by:

$$\rho((\boldsymbol{\nu}x)(P \mid Q)) = \rho'(P\{\rho(x)/x\}) \mid \rho'(Q\{\overline{\rho(x)}/x\})$$

where $\rho' = \rho \setminus \{x\}$ and $\overline{\rho(x)}$ denotes $s[\mathsf{q}][\mathsf{p}]$ if $\rho(x) = s[\mathsf{p}][\mathsf{q}]$ (with the congruence cases).

Examples of the translation. We give three examples of the translation from CLL to MP.

Example 3.3 (Conditions of comp*).* We explain the conditions of comp via a small example. Consider the following processes:

$$P \triangleq x\langle 7\rangle.z\langle\text{"}hello\text{"}\rangle.\mathbf{0} \quad Q_1 \triangleq z(u).y(w).\mathbf{0} \quad Q_2 \triangleq z(v).\mathbf{0} \mid y(w).\mathbf{0}$$
$$P \vdash_{\mathsf{CL}} z{:}\mathsf{str} \otimes \mathbf{1}, x{:}\mathsf{nat} \otimes \mathbf{1} \quad Q_i \vdash_{\mathsf{CL}} z{:}\mathsf{str} \parr \bot, y{:}\mathsf{nat} \parr \bot$$

We define σ, σ_1 and σ_2 such that:

$$\begin{array}{lcl} \sigma(P) & = & s[\mathsf{p}][\mathsf{r}]\langle 7\rangle; s[\mathsf{p}][\mathsf{q}]\langle\text{"}hello\text{"}\rangle \\ \sigma_1(Q_1) & = & s[\mathsf{q}][\mathsf{p}](u); s[\mathsf{r}_1][\mathsf{r}_2](w) \\ \sigma_2(Q_2) & = & s[\mathsf{q}][\mathsf{p}](v) \mid s[\mathsf{r}_1][\mathsf{r}_2](w) \end{array}$$

Then, assuming $\rho = \{\}$, the mappings σ and σ_i above satisfy **(b)** $(z, s[\mathsf{p}][\mathsf{q}]) \in \sigma$ and $(z, s[\mathsf{q}][\mathsf{p}]) \in \sigma_i$ by $\mathsf{p} = \mathsf{p}_\sigma(z) = \mathsf{d}_{\sigma_i}(z)$ and $\mathsf{q} = \mathsf{d}_\sigma(z) = \mathsf{p}_{\sigma_i}(z)$; **(c)** $(x, s[\mathsf{p}][\mathsf{r}]) \in \sigma$ and $(y, s[\mathsf{r}_1][\mathsf{r}_2]) \in \sigma_i$ with $\mathsf{r}_1 = \mathsf{q}$ in σ_1 and $\mathsf{r}_1 \neq \mathsf{q}$ in σ_2; and $\mathsf{d}_\sigma(x) = \mathsf{r} \neq \mathsf{r}_2 = \mathsf{d}_{\sigma_i}(y)$; and $\mathsf{r}, \mathsf{r}_2 \notin \rho$.

Example 3.4 (Four Threads). We show how to translate and compose the CLL process P from the beginning of this section:

$$\begin{array}{lcl} P & \triangleq & x\langle 7\rangle.y(z).x\langle\text{"}hello\text{"}\rangle.\mathbf{0} \vdash_{\mathsf{CL}} x{:}\mathsf{nat} \otimes \mathsf{str} \otimes \mathbf{1}, y{:}\mathsf{nat} \parr \bot \\ Q_1 & \triangleq & x(x_1).w\langle 93\rangle.x(x_2).\mathbf{0} \vdash_{\mathsf{CL}} x{:}\mathsf{nat} \parr \mathsf{str} \parr \bot, w{:}\mathsf{nat} \otimes \mathbf{1} \\ Q_2 & \triangleq & y\langle 2\rangle.\mathbf{0} \vdash_{\mathsf{CL}} y{:}\mathsf{nat} \otimes \mathbf{1} \quad Q_3 \triangleq w(x_3).\mathbf{0} \vdash_{\mathsf{CL}} w{:}\mathsf{nat} \parr \bot \end{array}$$

We define $\sigma, \sigma_1, \sigma_2, \sigma_3$ such that:

$$\begin{array}{lcl@{\qquad}lcl} \sigma(P) & = & s[\mathsf{p}][\mathsf{q}]\langle 7\rangle; s[\mathsf{p}][\mathsf{r}](z); s[\mathsf{p}][\mathsf{q}]\langle\text{"}hello\text{"}\rangle; \mathbf{0} & [\![P]\!]_\sigma & = & \mathsf{q}\uparrow(\mathsf{nat}); \mathsf{r}\downarrow(\mathsf{nat}); \mathsf{q}\uparrow(\mathsf{str}); \mathsf{end} \\ \sigma_1(Q_1) & = & s[\mathsf{q}][\mathsf{p}](x_1); s[\mathsf{q}][\mathsf{s}]\langle 93\rangle; s[\mathsf{q}][\mathsf{p}](x_2); \mathbf{0} & [\![Q_1]\!]_{\sigma_1} & = & \mathsf{p}\downarrow(\mathsf{nat}); \mathsf{s}\uparrow(\mathsf{nat}); \mathsf{p}\downarrow(\mathsf{str}); \mathsf{end} \\ \sigma_2(Q_2) & = & s[\mathsf{r}][\mathsf{p}]\langle 2\rangle; \mathbf{0} & [\![Q_2]\!]_{\sigma_2} & = & \mathsf{p}\uparrow(\mathsf{nat}); \mathsf{end} \\ \sigma_3(Q_3) & = & s[\mathsf{s}][\mathsf{q}](x_3); \mathbf{0} & [\![Q_3]\!]_{\sigma_3} & = & \mathsf{q}\downarrow(\mathsf{nat}); \mathsf{end} \end{array}$$

Let $\Gamma = s[\mathsf{p}]{:}[\![P]\!]_\sigma, s[\mathsf{q}]{:}[\![Q_1]\!]_{\sigma_1}, s[\mathsf{r}]{:}[\![Q_2]\!]_{\sigma_2}, s[\mathsf{s}]{:}[\![Q_3]\!]_{\sigma_3}$. Then we have: $(\boldsymbol{\nu}x, y, w)(P \mid Q_1 \mid Q_2 \mid Q_3) \Vdash^{\emptyset}_{\rho} \cdot; \Gamma$.

Example 3.5 (Choice and Branching). As we discuss in § 4, this CLL typable branching behaviour is not typable in the previous work on multiparty logic [11, 14] using the same local types:

$$\begin{array}{l@{\qquad}l} P \triangleq x.\mathsf{case}\{l_1{:}y.l_2; \mathbf{0}, l_3{:}y.l_4; \mathbf{0}\} & Q_1 \triangleq x.l_1; \mathbf{0} \\ R \triangleq y.\mathsf{case}\{l_2{:}\mathbf{0}, l_4{:}\mathbf{0}\} & Q_2 \triangleq x.l_3; \mathbf{0} \end{array}$$

with: $P \vdash_{\mathsf{CL}} x{:} \,\&\, \{l_1{:}\bot, l_3{:}\bot\}, y{:} \oplus \{l_2{:}\mathbf{1}, l_4{:}\mathbf{1}\}$, $R \vdash_{\mathsf{CL}} y{:} \,\&\, \{l_2{:}\bot, l_4{:}\bot\}$ and $Q_i \vdash_{\mathsf{CL}} x{:} \oplus \{l_1{:}\mathbf{1}, l_3{:}\mathbf{1}\}$. We define mappings σ, σ_1 and σ_2 such that:

$$
\begin{array}{lcl}
\sigma(P) & = & s[\mathsf{p}][\mathsf{q}] \,\&\, \{l_1{:}s[\mathsf{p}][\mathsf{r}] \oplus l_2; \mathbf{0}, l_3{:}s[\mathsf{p}][\mathsf{r}] \oplus l_4; \mathbf{0}\} \\
\sigma_1(Q_1) & = & s[\mathsf{q}][\mathsf{p}] \oplus l_1; \mathbf{0} \\
\sigma_1(Q_2) & = & s[\mathsf{q}][\mathsf{p}] \oplus l_3; \mathbf{0} \\
\sigma_2(R) & = & s[\mathsf{r}][\mathsf{p}] \,\&\, \{l_2{:}\mathbf{0}, l_4{:}\mathbf{0}\} \\
[\![P]\!]_\sigma & = & \&\mathsf{q}\{l_1{:} \oplus \mathsf{r}\{l_2{:}\mathsf{end}\}, l_3{:} \oplus \mathsf{r}\{l_4{:}\mathsf{end}\}\} \\
[\![Q_1]\!]_{\sigma_1} & = & \oplus\mathsf{p}\{l_1{:}\mathsf{end}\} \\
[\![Q_2]\!]_{\sigma_1} & = & \oplus\mathsf{p}\{l_3{:}\mathsf{end}\} \\
[\![R]\!]_{\sigma_2} & = & \&\mathsf{p}\{l_2{:}\mathsf{end}, l_4{:}\mathsf{end}\}
\end{array}
$$

Let $\Gamma = s[\mathsf{p}]{:}[\![P]\!]_\sigma, s[\mathsf{q}]{:}[\![Q_i]\!]_{\sigma_1}, s[\mathsf{r}]{:}[\![R]\!]_{\sigma_2}$. Thus we have: $(\boldsymbol{\nu} x, y)(P \mid Q_i \mid R) \Vdash^{\emptyset}_{\rho} \cdot; \Gamma$.

Type-preservation and uniqueness. Below we study properties of the encoding. We first show that the type preserving translation of CLL to MP for cut-free processes combined with our composition rule preserves typing in MP.

Proposition 3.6 (Type Preservation). *If $P \Vdash^{\sigma}_{\rho} \Delta; \Gamma$, then $\rho(\sigma(P)) \vdash_{\mathsf{MP}} \Gamma$.*

Proof. The prefix case is straightforward by Definition 3.1 and (thread); the parallel composition uses (comp). Both cases are mechanical by induction on P. □

Since the mapping from CLL into MP is just renaming, reduction of CLL strongly corresponds to that of MP.

Proposition 3.7 (Operational Correspondence). *Suppose $P \Vdash^{\sigma}_{\rho} \Delta; \Gamma$ and $P \to P'$. Then $\rho(\sigma(P)) \to Q$ s.t. $P' \Vdash^{\sigma'}_{\rho'} \Delta'; \Gamma'$ and $Q = \rho'(\sigma'(P'))$ with $\sigma' \subseteq \sigma$, $\rho' \subseteq \rho$.*

Proof. See Appendix A.1.1. □

We call a mapping *thread-preserving* if it assigns to a cut-free CLL process a single participant in MP. We thus have:

Proposition 3.8 (Thread Preservation). *If $P \Vdash^{\sigma}_{\rho} \Delta; \Gamma$, then $\rho(\sigma(P))$ is thread-preserving.*

Proof. See Appendix A.1.1. □

Theorem 3.9 states that the mapping is closed under any bijective renaming φ on sessions, roles and channels. As an example, let $P = s[\mathsf{p}][\mathsf{r}](x); s[\mathsf{p}][\mathsf{q}]\langle v\rangle$ and $\varphi = \{s \mapsto s', x \mapsto y, \mathsf{p} \mapsto \mathsf{p}'\}$; then $\varphi(P) = s'[\mathsf{p}'][\mathsf{r}](y); s'[\mathsf{p}'][\mathsf{q}]\langle v\rangle$.

More precisely, Theorem 3.9 shows that any thread-preserving mapping from CLL processes into a single MP session always conforms to our mapping. This means that no other way to encode CLL into MP (modulo bijective renaming which maps different names to distinct destinations) exists if it is thread-preserving into a single multiparty session.

Theorem 3.9 (Uniqueness). *Assume $P \vdash_{\mathsf{CL}} \Delta$. Suppose $\varphi(P)$ is thread-preserving and $\varphi(P)$ is typable by a single MP session, i.e. if $\varphi(P) \vdash_{\mathsf{MP}} \Gamma$ then (1) $\mathrm{dom}(\Gamma)$ contains a single session channel; or (2) $\Gamma = \emptyset$ and $P \equiv \mathbf{0}$. Then there exist ρ and σ such that $\varphi = \sigma \circ \rho$ and $P \Vdash^{\sigma}_{\rho} \Delta; \Gamma$.*

Proof. See Appendix A.1. □

4 PARTIAL MULTIPARTY COMPATIBILITY

This section studies a compositional synthesis property, dubbed *partial multiparty compatibility* (PMC). As illustrated in Proposition 2.7, multiparty session type theories [31] cannot ensure deadlock-freedom if we do not rely on (1) a projection from a global type; or (2) a global synthesis property called multiparty compatibility [22, 37], which given local types for a *complete* MP session produces a global type if the endpoints do not deadlock. For example, the previous counterexample can be avoided if we start from the global type (again, we assume basic value passing, encodable with terminated sessions):

$$G = \mathsf{p} \to \mathsf{q}{:}(\mathsf{nat}).\mathsf{q} \to \mathsf{r}{:}(\mathsf{bool}).\mathsf{r} \to \mathsf{p}{:}(\mathsf{str}).\mathsf{end} \tag{2}$$

and type each process with the projected local types:

$$T_{\mathsf{p}} = \mathsf{q}{\uparrow}(\mathsf{nat}); \mathsf{r}{\downarrow}(\mathsf{str}); \mathsf{end}, \quad T_{\mathsf{q}} = \mathsf{p}{\downarrow}(\mathsf{nat}); \mathsf{r}{\uparrow}(\mathsf{bool}); \mathsf{end}, \quad T_{\mathsf{r}} = \mathsf{q}{\downarrow}(\mathsf{bool}); \mathsf{p}{\uparrow}(\mathsf{str}); \mathsf{end} \tag{3}$$

or we may build (synthesise) G in (2) from $\{T_{\mathsf{p}}, T_{\mathsf{q}}, T_{\mathsf{r}}\}$ in (3). If we start from a projectable global type or can synthesise a global type, the example in Proposition 2.7 is no longer typable.

Given that CLL employs a binary form of composition, we move from a global synthesis condition to a binary (partial) relation to achieve our main results. Specifically, we take the following steps:

Step 1: We introduce *partial* global types $\mathsf{p} \rightsquigarrow \mathsf{q}$ representing 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), and give formal semantics to both global and local types (§ 4.1) as labelled transition systems. Crucially, the semantics of global types is given up to a swapping relation $\sim_{\mathsf{sw}}$, which enables the permutation of independent actions.

Step 2: We define synchronous multiparty compatibility (SMC – Definition 4.7), showing the equivalence of SMC, deadlock-freedom and the existence of a global type that corresponds to the appropriate local behaviours.

Step 3: We introduce a notion of fusion (Definition 4.10), which enables us to compose compatible partial specifications and define *partial* multiparty compatibility. When a $\mathsf{p} \rightsquigarrow \mathsf{q}$ arrow denoting a send action is fused with the corresponding arrow denoting the receive action, it is transformed into a complete arrow $\mathsf{p} \to \mathsf{q}$, preserving the ordering of communications. When we compose all participants in a session (reconstructing a *complete* global type – one without partial arrows), deadlock-freedom is guaranteed (Theorem 4.15).

4.1 Partial Global Types and Semantics

We define partial global types 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' is:

$$\begin{array}{rcl} G & ::= & \mathsf{end} \mid \mathsf{p} \to \mathsf{q}{:}(T).G \mid \mathsf{p} \to \mathsf{q}{:}\{l_j{:}G_j\}_{j \in J} \\ & \mid & \mathsf{p} \rightsquigarrow \mathsf{q}{:} \uparrow (T).G \mid \mathsf{p} \rightsquigarrow \mathsf{q}{:} \downarrow (T).G \mid \mathsf{p} \rightsquigarrow \mathsf{q}{:}\oplus\{l_j{:}G_j\}_{j \in J} \mid \mathsf{p} \rightsquigarrow \mathsf{q}{:} \,\&\, \{l_j{:}G_j\}_{j \in J} \end{array}$$

The first three of the above grammar constructs are the standard global types [31]. Global type $\mathsf{p} \to \mathsf{q}{:}(T).G$ means that participant p sends a session endpoint of type T to participant q, followed by G. Global type $\mathsf{p} \to \mathsf{q}{:}\{l_j{:}G_j\}_{j \in J}$ means that participant p selects label l_i, then q's i-th branch will be chosen, becoming G_i. The *partial* global types in the second line denote *half* of a complete global interaction. The modes $(\uparrow, \downarrow, \oplus, \&)$ in partial global types indicate which component of the interaction is being satisfied: e.g. $\mathsf{p} \rightsquigarrow \mathsf{q}{:} \uparrow (T)$ denotes the contribution of the emission component of the interaction from principal p to destination q, whereas $\mathsf{p} \rightsquigarrow \mathsf{q}{:} \downarrow (T)$ denotes the reception.

We write mode $\dagger$ for either $\uparrow, \downarrow, \oplus, \&$ or $\emptyset$ (empty) and often omit $\dagger$ from partial global types when unnecessary. We write $\rightarrowtriangle$ for either $\to$ or $\rightsquigarrow$; and $\mathsf{p} \leftrightarrowtriangle \mathsf{q}$ for either $\mathsf{p} \rightarrowtriangle \mathsf{q}$ or $\mathsf{q} \rightarrowtriangle \mathsf{p}$.

The set of *principal roles* is defined as: $\mathsf{pr}(\mathsf{p} \to \mathsf{q} : \emptyset) = \{\mathsf{p}, \mathsf{q}\}$ and $\mathsf{pr}(\mathsf{p} \rightsquigarrow \mathsf{q} :\uparrow) = \mathsf{pr}(\mathsf{q} \rightsquigarrow \mathsf{p} :\downarrow) = \mathsf{pr}(\mathsf{p} \rightsquigarrow \mathsf{q} : \oplus) = \mathsf{pr}(\mathsf{q} \rightsquigarrow \mathsf{p} : \&) = \{\mathsf{p}\}$. We write $\mathsf{roles}(G)/\mathsf{pr}(G)$ for the set of roles/principal roles occurring in G; and $\mathsf{p} \twoheadleftarrow\!\!\!\twoheadrightarrow \mathsf{q} \in G$ if $\mathsf{p} \twoheadleftarrow\!\!\!\twoheadrightarrow \mathsf{q}$ occurs in G.

We use standard projection rules from global to local types, defined in terms of a merge operation for branchings [22], 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 label (the operator is otherwise undefined).

Definition 4.2 (Local Type Merge). The *merge* $T \sqcup T'$ of T and T' is defined by $T \sqcup T \triangleq T$; and with $T = \&\mathsf{r}\{\mathsf{l}_i : T_i\}_{i\in I}$ and $T' = \&\mathsf{r}\{\mathsf{l}'_j : T'_j\}_{j\in J}$,

$$T \sqcup T' \triangleq \&\mathsf{r}(\{\mathsf{l}_h : T_h\}_{h\in I\setminus J} \cup \{\mathsf{l}'_h : T'_h\}_{h\in J\setminus I} \cup \{\mathsf{l}_h : T_h \sqcup T'_h\}_{h\in I\cap J})$$

if $\mathsf{l}_h = \mathsf{l}'_h$ for each $h \in I \cap J$; and homomorphic for other types (i.e. $\mathcal{T}[T_1] \sqcup \mathcal{T}[T_2] = \mathcal{T}[T_1 \sqcup T_2]$ where $\mathcal{T}$ is a context of local types). $T \sqcup T'$ is undefined otherwise. ◇

Definition 4.3 (Projection and Well-formedness). Let G be a global type. The projection of G for a role p, written $G \upharpoonright \mathsf{p}$, is defined below.

$$\begin{aligned}
\mathsf{end} \upharpoonright \mathsf{p} \quad &= \quad \mathsf{end}\\
\mathsf{s} \to \mathsf{r}{:}(T).G' \upharpoonright \mathsf{p} \quad &= \quad \begin{cases} \mathsf{r}\!\uparrow\!(T); (G' \upharpoonright \mathsf{p}) & \text{if } \mathsf{p} = \mathsf{s}\\ \mathsf{s}\!\downarrow\!(T); (G' \upharpoonright \mathsf{p}) & \text{if } \mathsf{p} = \mathsf{r}\\ G' \upharpoonright \mathsf{p} & \text{otherwise}\end{cases}\\
\mathsf{s} \to \mathsf{r}{:}\{l_j{:}G_j\}_{j\in J} \upharpoonright \mathsf{p} \quad &= \quad \begin{cases} \oplus\mathsf{r}\{l_j{:}G_j \upharpoonright \mathsf{p}\}_{j\in J} & \text{if } \mathsf{p} = \mathsf{s}\\ \&\mathsf{s}\{l_j{:}G_j \upharpoonright \mathsf{p}\}_{j\in J} & \text{if } \mathsf{p} = \mathsf{r}\\ \sqcup_{j\in J} G_j \upharpoonright \mathsf{p} & \text{otherwise}\end{cases}
\end{aligned}$$

If no side conditions hold (i.e. the merge operator is undefined) then projection is undefined. We say that G is *well-formed* iff for all distinct $\mathsf{p} \in \mathsf{roles}(G)$, $(G{\upharpoonright}\mathsf{p})$ is defined.

As an illustration of merge and projection, consider

$$G = \mathsf{q} \to \mathsf{p}{:}\{l_1{:}\mathsf{p} \to \mathsf{r}{:}\{l_2{:}\mathsf{end}\}, l_3{:}\mathsf{p} \to \mathsf{r}{:}\{l_4{:}\mathsf{end}\}\}$$

which will be built from CLL processes in Example 3.5. Then:

$$\begin{gathered}
G{\upharpoonright}\mathsf{p} = \&\mathsf{q}\{l_1{:} \oplus \mathsf{r}\{l_2 : \mathsf{end}\}, l_3{:} \oplus \mathsf{r}\{l_4 : \mathsf{end}\}\},\\
G{\upharpoonright}\mathsf{q} = \oplus\mathsf{p}\{l_1 : \mathsf{end}, l_3 : \mathsf{end}\} \text{ and } G{\upharpoonright}\mathsf{r} = \&\mathsf{p}\{l_2{:}\mathsf{end}, l_4{:}\mathsf{end}\}.
\end{gathered} \tag{4}$$

The syntax of global types can impose unnecessary orderings of actions among independent roles. For example, $\mathsf{p} \twoheadrightarrow \mathsf{q}{:}(\mathsf{str}).\mathsf{r} \twoheadrightarrow \mathsf{s}{:}\ :(\mathsf{bool}).\mathsf{end}$ should be regarded as identical to $\mathsf{r} \twoheadrightarrow \mathsf{s}{:}(\mathsf{bool}).\mathsf{p} \twoheadrightarrow \mathsf{q}{:}(\mathsf{str}).\mathsf{end}$ if p and q do not coincide with either r or s, since there is no reasonably enforceable ordering between the two interactions. Thus, we allow for the swapping of independent communication actions in global types as defined below (a similar swapping relation is used [14] to define coherence of logical global types and in the context of semantics for choreographies [12]).

Definition 4.4 (Swapping). We define the swapping relation, written $\sim_{\mathsf{sw}}$, as the smallest congruence on global types satisfying (with $\mathsf{pr}(\mathsf{p} \twoheadrightarrow \mathsf{q} : \dagger) \cap \mathsf{pr}(\mathsf{p}' \twoheadrightarrow \mathsf{q}' : \dagger') = \emptyset$):

$$\begin{array}{llll}
\text{(ss)} & \mathsf{p} \twoheadrightarrow \mathsf{q}{:} \dagger (T).\mathsf{p}' \twoheadrightarrow \mathsf{q}'{:} \dagger' (T').G & \sim_{\mathsf{sw}} & \mathsf{p}' \twoheadrightarrow \mathsf{q}'{:} \dagger' (T').\mathsf{p} \twoheadrightarrow \mathsf{q}{:} \dagger (T).G\\
\text{(sb)} & \mathsf{p} \twoheadrightarrow \mathsf{q}{:} \dagger (T).\mathsf{p}' \twoheadrightarrow \mathsf{q}'{:} \dagger' \{l_i : G_i\}_{i\in I} & \sim_{\mathsf{sw}} & \mathsf{p}' \twoheadrightarrow \mathsf{q}'{:} \dagger' \{l_i : \mathsf{p} \twoheadrightarrow \mathsf{q}{:} \dagger (T).G_i\}_{i\in I}\\
\text{(bb)} & \mathsf{p} \twoheadrightarrow \mathsf{q}{:} \dagger \{l_i : \mathsf{p}' \twoheadrightarrow \mathsf{q}'{:} \dagger' \{l'_j : G_j\}_{j\in J}\}_{i\in I} & \sim_{\mathsf{sw}} & \mathsf{p}' \twoheadrightarrow \mathsf{q}'{:} \dagger' \{l'_j : \mathsf{p} \twoheadrightarrow \mathsf{q}{:} \dagger \{l_i : G_j\}_{i\in I}\}_{j\in J}
\end{array}$$

Local Type (at role p***)***

$$\mathsf{q}\uparrow(S);T \xrightarrow{\mathsf{pq}\uparrow(S)} T \quad \mathsf{q}\downarrow(S);T \xrightarrow{\mathsf{pq}\downarrow(S)} T \quad \oplus\mathsf{q}\{\mathsf{l}_i{:}T_i\}_{i\in I} \xrightarrow{\mathsf{pq}\triangleleft l_k} T_k\ (k\in I) \quad \&\mathsf{q}\{\mathsf{l}_i{:}T_i\}_{i\in I} \xrightarrow{\mathsf{pq}\triangleright l_k} T_k\ (k\in I)$$

Global Type

$$\mathsf{p}\to\mathsf{q}{:}(T).G \xrightarrow{\mathsf{pq}\uparrow(T)\cdot\mathsf{qp}\downarrow(T)} G \qquad \mathsf{p}\to\mathsf{q}{:}\{l_i : G_i\}_{i\in I} \xrightarrow{\mathsf{pq}\triangleleft l_k\cdot\mathsf{qp}\triangleright l_k} G_k\ (k\in I)$$

$$G_1 \sim_{\mathsf{sw}} G_1' \wedge G_1' \xrightarrow{\ell\cdot\ell'} G_2' \wedge G_2' \sim_{\mathsf{sw}} G_2 \;\Rightarrow\; G_1 \xrightarrow{\ell\cdot\ell'} G_2$$

Configuration

$$(T_{\mathsf{p}} \xrightarrow{\ell} T_{\mathsf{p}}') \wedge (T_{\mathsf{q}} \xrightarrow{\overline{\ell}} T_{\mathsf{q}}') \wedge (\forall \mathsf{r}\in\mathcal{P}\setminus\{\mathsf{p},\mathsf{q}\}.\, T_{\mathsf{r}} = T_{\mathsf{r}}') \Rightarrow (T_{\mathsf{p}})_{\mathsf{p}\in\mathcal{P}} \xrightarrow{\ell\cdot\overline{\ell}} (T_{\mathsf{p}}')_{\mathsf{p}\in\mathcal{P}} \quad (\ell \text{ output or selection})$$

Fig. 5. Labelled Transition Systems

The operational semantics for local, global types and configurations are given by labelled transition systems (LTS). We define the syntax of labels as:

$$\ell ::= \mathsf{pq}\uparrow(T) \mid \mathsf{pq}\downarrow(T) \mid \mathsf{pq}\triangleleft l \mid \mathsf{pq}\triangleright l$$

where $\mathsf{pq}\uparrow(T)$ (resp. $\mathsf{pq}\downarrow(T)$) is the output (resp. input) at p to q (resp. from q) and $\mathsf{pq}\triangleleft l$ (resp. $\mathsf{pq}\triangleright l$) is the selection (resp. branching).

We define $\overline{\ell}$ as $\overline{\mathsf{pq}\uparrow(\tau)} = \mathsf{qp}\downarrow(\tau)$ and $\overline{\mathsf{pq}\triangleleft l} = \mathsf{qp}\triangleright l$ and vice-versa. Given a set of roles $\mathcal{P}$, we define *a configuration* as $C = (T_{\mathsf{p}})_{\mathsf{p}\in\mathcal{P}}$. Configurations consist of a set of local types projected from a single global type which are used to define and show properties of local types in § 4.2.

Definition 4.5 (Labelled Transition Relations). Transitions between local types, written $T \xrightarrow{\ell} T'$, for role p; global types, written $G \xrightarrow{\ell\cdot\ell'} G'$; and configurations are given in Fig. 5. We write $G \xrightarrow{\vec{\ell}} G_n$ if $G \xrightarrow{\ell_1\cdot\ell_2} G_1 \cdots \xrightarrow{\ell_{2n-1}\cdot\ell_{2n}} G_n$ and $\vec{\ell} = \ell_1\cdots\ell_{2n}$ $(n \geq 0)$; and $Tr(G_0) = \{\vec{\ell} \mid G_0 \xrightarrow{\vec{\ell}} G_n\ n \geq 0\}$ for *traces* of type G_0. Similarly for T and C.

Proposition 4.6 (Trace Equivalence). *Suppose G well-formed and the set of participants in G is $\mathcal{P}$. Assume $C = (G\lceil\mathsf{p})_{\mathsf{p}\in\mathcal{P}}$. Then $Tr(C) = Tr(G)$.*

Proof. By definition of the projection and the LTSs. □

4.2 Partial Multiparty Compatibility (PMC)

To introduce PMC, we first define multiparty compatibility (MC) for a *synchronous* semantics, adapting the development of MC for asynchrony [22, 37]. We then introduce PMC as a compositional binary synthesis property on types. We note that while asynchronous formulations of the linear logic-based session calculi exist [23], the predominant formulation is synchronous, and so we focus on a synchronous theory.

Definition 4.7 (Synchronous Multiparty Compatibility). Configuration $C_0 = (T_{0\mathsf{p}})_{\mathsf{p}\in\mathcal{P}}$ is *synchronous multiparty compatible* (SMC) if for all $C_0 \xrightarrow{\vec{\ell}} C = (T_{\mathsf{p}})_{\mathsf{p}\in\mathcal{P}}$ and $T_{\mathsf{p}} \xrightarrow{\ell} T_{\mathsf{p}}'$:

(1) if $\ell = \mathsf{pq}\uparrow(S)$ or $\mathsf{pq}\triangleleft l$, there exists $C \xrightarrow{\vec{\ell}'} C' \xrightarrow{\ell\cdot\overline{\ell}} C''$;

(2) if $\ell = \mathsf{pq}\downarrow(S)$, there exists $C \xrightarrow{\vec{\ell}'} C' \xrightarrow{\overline{\ell}\cdot\ell} C''$; or

(3) if $\ell = \mathsf{pq}\triangleright l$, there exists $\ell_1 = \mathsf{pq}\triangleright l'$, $C \xrightarrow{\vec{\ell}'} C' \xrightarrow{\ell_1\cdot\overline{\ell}_1} C''$

where $\vec{\ell}'$ does not include actions from or to p.

Since our semantics is synchronous, it is technically simpler than one for an asynchronous semantics [22, 37]. One can check that the local types in Equation 3 and the projected local types of (4.1) satisfy SMC.

Definition 4.8 (Deadlock-freedom). $C = (T_{\mathrm{p}})_{\mathrm{p}\in\mathcal{P}}$ is *deadlock-free* if for all $C \xrightarrow{\vec{\ell}} C_1$, $\exists C' = (T'_{\mathrm{p}})_{\mathrm{p}\in\mathcal{P}}$ such that $C_1 \xrightarrow{\ell'} C'$ or $T'_{\mathrm{p}} = \mathsf{end}$ for all $\mathrm{p} \in \mathcal{P}$.

Theorem 4.9 (Deadlock-freedom, MC and existence of a global type). *The following are equivalent:* (MC) *A configuration C is SMC;* (DF) *C is deadlock-free;* (WF) *There exists well-formed G such that $Tr(G) = Tr(C)$.*

Proof. See Appendix A.2.1. □

Multiparty compatibility is a global property defined using the set of all participants [22, 37]. To define a compositional (i.e. local) multiparty compatibility, we introduce the composition of two partial global types, dubbed as *fusion*.

Definition 4.10 (Fusion). We define the fusion of two well-formed partial global types G_1, G_2 such that $\mathsf{pr}(G_1) \cap \mathsf{pr}(G_2) = \emptyset$, written $\mathsf{fuse}(G_1, G_2)$, inductively on the structure of G_1 and G_2, up to the swapping relation $\sim_{\mathrm{sw}}$:

$$\mathsf{fuse}(\mathrm{p} \rightsquigarrow \mathrm{q}{:}\uparrow(T_1).G'_1, \mathrm{p} \rightsquigarrow \mathrm{q}{:}\downarrow(T_2).G'_2) = \mathrm{p} \to \mathrm{q}{:}(T_2).\mathsf{fuse}(G'_1, G'_2) \quad (\text{with } T_1 \geq T_2)$$

$$\mathsf{fuse}(\mathrm{p} \rightsquigarrow \mathrm{q}{:}\oplus\{l : G'_1\}, \mathrm{p} \rightsquigarrow \mathrm{q}{:}\,\&\,\{l : G'_2, \{l_j : G_j\}_{j\in J}\}) = \mathrm{p} \to \mathrm{q}{:}\{l : \mathsf{fuse}(G'_1, G'_2)\}$$

$$\mathsf{fuse}(\mathrm{p} \rightsquigarrow \mathrm{q}{:}\dagger(T).G_1, G_2) = \mathrm{p} \rightsquigarrow \mathrm{q}{:}\dagger(T).\mathsf{fuse}(G_1, G_2)$$
$$\text{if } \nexists G'_2.(\mathrm{p} \rightsquigarrow \mathrm{q}{:}\overline{\dagger}(T').G'_2 \sim_{\mathrm{sw}} G_2) \wedge \mathrm{p} \leftrightarrow \mathrm{q} \notin G_2$$

$$\mathsf{fuse}(\mathrm{p} \rightsquigarrow \mathrm{q}{:}\dagger\{l_j : G_j\}_{j\in J}, G_2) = \mathrm{p} \rightsquigarrow \mathrm{q}{:}\dagger\{l_j : \mathsf{fuse}(G_j, G_2)\}_{j\in J}$$
$$\text{if } \nexists G'_j.(\mathrm{p} \rightsquigarrow \mathrm{q}{:}\overline{\dagger}\{l_j : G'_j\}_{j\in J} \sim_{\mathrm{sw}} G_2 \wedge i \in J) \wedge \mathrm{p} \leftrightarrow \mathrm{q} \notin G_2$$

$$\mathsf{fuse}(\mathrm{p} \to \mathrm{q} : (T).G_1, G_2) = \mathrm{p} \to \mathrm{q} : (T).\mathsf{fuse}(G_1, G_2) \text{ if } \mathrm{p} \leftrightarrow \mathrm{q} \notin G_2$$

$$\mathsf{fuse}(\mathrm{p} \to \mathrm{q} : \{l_j : G_j\}_{j\in J}, G_2) = \mathrm{p} \to \mathrm{q} : \{l_j : \mathsf{fuse}(G_j, G_2)\}_{j\in J} \text{ if } \mathrm{p} \leftrightarrow \mathrm{q} \notin G_2$$

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

with the symmetric cases.

The first rule uses the subtyping relation $T \leq S$ given in Definition 2.5. The second rule selects one branch with the same label. For simplicity, we allow only for one-way selections since in the context of our work, partial types are extracted from processes where the selections are always determined. We note that multi-way branchings can be realised straightforwardly via subtyping.

The third and forth rules (which do not overlap with the first two) push through actions that are unmatched in the fused types. The rule is extended similarly to input, branching and selection with other global type constructors. $\mathsf{fuse}(G_1, G_2)$ is undefined if none of the above rules are applicable.

We define (1) $|\mathsf{end}| = 1$; (2) $|\mathrm{p} \rightarrow \mathrm{q} : \dagger(T).G| = 4 + |G|$; and (3) $|\mathrm{p} \rightarrow \mathrm{q} : \dagger\{l_j{:}G_j\}_{j\in J}| = 3 + \Sigma_{j\in J}(1 + |G_j|)$. We have:

Proposition 4.11. *Computing* $\mathsf{fuse}(G_1, G_2)$ *is* $\mathcal{O}(|G_1|! \times |G_2|!)$ *time in the worst case, where $|G|$ is the size of G.*

Proof. The time complexity is dominated by the computation of the swapping relation between G_1 and G_2. The equivalence class up to the swapping relation consists of $|G|!$ elements. Since we apply each fuse rule to the equivalence class of G_1 and the equivalence class of G_2 (in the worst case), the time complexity is $\mathcal{O}(|G_1|! \times |G_2|!)$. □

Definition 4.12 (Partial Multiparty Compatibility). Suppose G_1 and G_2 are partial global types. G_1 and G_2 are partial multiparty compatible iff $\mathsf{fuse}(G_1, G_2)$ is defined.

Example 4.13. Consider the following partial global types for Example 3.5 (as mentioned before, the corresponding local types are not coherent in [11, 14]):

$$
\begin{array}{l}
G_1 = \mathsf{q} \rightsquigarrow \mathsf{p}{:}\ \&\ \{l_1{:}\mathsf{p} \rightsquigarrow \mathsf{r}{:}\oplus\{l_2{:}\mathsf{end}\}, l_3{:}\mathsf{p} \rightsquigarrow \mathsf{r}{:}\oplus\{l_4{:}\mathsf{end}\}\}\\
G_2 = \mathsf{q} \rightsquigarrow \mathsf{p}{:}\oplus\{l_1{:}\mathsf{end}\}\\
G_3 \ = \ \mathsf{p} \rightsquigarrow \mathsf{r}{:}\ \&\ \{l_2{:}\mathsf{end}, l_4{:}\mathsf{end}\}\\
G_4 = \mathsf{q} \rightsquigarrow \mathsf{p}{:}\oplus\{l_3{:}\mathsf{end}\}
\end{array}
$$

Then we have:

$$
\begin{array}{l}
\mathsf{fuse}(G_1, G_2) = \mathsf{q} \to \mathsf{p}{:}\{l_1{:}\mathsf{p} \rightsquigarrow \mathsf{r}{:}\oplus\{l_2{:}\mathsf{end}\}\}\\
\mathsf{fuse}(G_2, G_3) = \mathsf{q} \rightsquigarrow \mathsf{p}{:}\oplus\{l_1{:}G_3\}\\
\mathsf{fuse}(\mathsf{fuse}(G_1, G_2), G_3) = \mathsf{fuse}(G_1, \mathsf{fuse}(G_2, G_3)) = \mathsf{q} \to \mathsf{p}{:}\{l_1{:}\mathsf{p} \to \mathsf{r}{:}\{l_2{:}\mathsf{end}\}\}\\
\mathsf{fuse}(G_1, G_4) = \mathsf{q} \to \mathsf{p}{:}\{l_3{:}\mathsf{p} \rightsquigarrow \mathsf{r}{:}\oplus\{l_4{:}\mathsf{end}\}\}\\
\mathsf{fuse}(G_4, G_3) = \mathsf{q} \rightsquigarrow \mathsf{p}{:}\oplus\{l_3{:}G_3\}\\
\mathsf{fuse}(\mathsf{fuse}(G_1, G_4), G_3) = \mathsf{fuse}(G_1, \mathsf{fuse}(G_4, G_3)) = \mathsf{q} \to \mathsf{p}{:}\{l_3{:}\mathsf{p} \to \mathsf{r}{:}\{l_4{:}\mathsf{end}\}\}
\end{array}
$$

Lemma 4.14. *Suppose* $\mathsf{fuse}(\mathsf{fuse}(G_i, G_j), G_k)$ *with* $\{i, j, k\} = \{1, 2, 3\}$ *is well-formed. Then we have* $\mathsf{fuse}(\mathsf{fuse}(G_i, G_j), G_k) \sim_{\mathsf{sw}} \mathsf{fuse}(G_i, \mathsf{fuse}(G_j, G_k))$.

Proof. See Appendix A.2.2. □

By the above lemma, we have:

Theorem 4.15 (Compositionality). *Suppose* G_1, ...,G_n *are partial global types. Assume* $\forall i, j$ *such that* $1 \leq i \neq j \leq n$, G_i *and* G_j *are PMC and* $G = \mathsf{fuse}(G_1, \mathsf{fuse}(G_2\ , \mathsf{fuse}(\ldots, G_n)))$ *is a complete global type. Then* G *is well-formed.*

Proof. See Appendix A.2.3. □

5 ENCODING CLL AS A SINGLE MULTIPARTY SESSION

Having defined in § 3 how to translate CLL processes to MP, we study the interconnection networks induced by CLL by generating their partial global types (§ 4). We note that such types are well-formed by construction. We prove a strict inclusion of the networks 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 P with $P \vdash^{\sigma}_{\mathsf{CL}} \Delta$ we generate its partial global type, written $(\!|P|\!)_\sigma$ as follows (let $\mathsf{d}_\sigma(x) = \mathsf{q}$):

$$
\begin{array}{l}
(\!|\mathbf{0}|\!)_\sigma \triangleq \mathsf{end} \quad (\!|\overline{x}\langle y\rangle.P|\!)_\sigma \triangleq \mathsf{p}_\sigma \rightsquigarrow \mathsf{q}{:} \uparrow (T).(\!|P|\!)_\sigma \quad (\!|x(y).P|\!)_\sigma \triangleq \mathsf{q} \rightsquigarrow \mathsf{p}_\sigma{:} \downarrow (T).(\!|P|\!)_\sigma\\
(\!|x.l; P|\!)_\sigma \triangleq \mathsf{p}_\sigma \rightsquigarrow \mathsf{q}{:}\oplus\{l{:}(\!|P|\!)_\sigma\} \quad (\!|x.\mathsf{case}\{l_i : P_i\}_{i\in I}|\!)_\sigma \triangleq \mathsf{q} \rightsquigarrow \mathsf{p}_\sigma{:}\ \&\ \{l_i{:}(\!|P_i|\!)_\sigma\}_{i\in I}
\end{array}
$$

We generate a set $\mathcal{G}$ of partial global types for compositions, written $P \Vdash^{\sigma}_{\rho} \Delta; \mathcal{G}$, by (we omit the obvious (thread) rule):

$$
\frac{P \vdash^{\sigma}_{\mathsf{CL}} \Delta, x{:}A \quad Q \Vdash^{\sigma'}_{\rho} \Delta', x{:}A^{\perp}; \mathcal{G} \quad (\star) \text{ in (comp)}}{(\boldsymbol{\nu} x)(P \mid Q) \Vdash^{(\sigma' \cup \sigma)\setminus\{a\}}_{\rho'} \Delta, \Delta'; \mathcal{G} \cup (\!|P|\!)_\sigma}\ (\text{comp-}\mathcal{G})
$$

Example 5.2 (Four Threads). We present the generated partial global types for Example 3.4:

$$\begin{array}{rcl} (\!|P|\!)_\sigma & = & \mathsf{p} \rightsquigarrow \mathsf{q}{:}\uparrow(\mathsf{nat}).\mathsf{r} \rightsquigarrow \mathsf{p}{:}\downarrow(\mathsf{nat}).\mathsf{p} \rightsquigarrow \mathsf{q}{:}\uparrow(\mathsf{str}).\mathsf{end} \\ (\!|Q|\!)_{\sigma_2} & = & \mathsf{p} \rightsquigarrow \mathsf{q}{:}\downarrow(\mathsf{nat}).\mathsf{q} \rightsquigarrow \mathsf{s}{:}\uparrow(\mathsf{nat}).\mathsf{p} \rightsquigarrow \mathsf{q}{:}\downarrow(\mathsf{str}).\mathsf{end} \\ (\!|R|\!)_{\sigma_1} & = & \mathsf{r} \rightsquigarrow \mathsf{p}{:}\uparrow(\mathsf{nat}).\mathsf{end} \quad (\!|S|\!)_{\sigma_3} = \mathsf{q} \rightsquigarrow \mathsf{s}{:}\downarrow(\mathsf{nat}).\mathsf{end} \end{array}$$

Where applying fuse to the partial global types above produces the global type: $\mathsf{p} \to \mathsf{q}{:}(\mathsf{nat}).\mathsf{q} \to \mathsf{s}{:}(\mathsf{nat}).\mathsf{r} \to \mathsf{p}{:}(\mathsf{nat}).\mathsf{p} \to \mathsf{q}{:}(\mathsf{str}).\mathsf{end}$. We note that, for instance, adding a message from r to s makes the example untypable in CLL since it introduces a 3-way cycle in the interconnection network.

Example 5.3 (Choice and Branching). The partial global types generated for Example 3.5 are $G_1 = (\!|P|\!)_\sigma, G_2 = (\!|Q_1|\!)_{\sigma_1}, G_3 = (\!|R|\!)_{\sigma_2}$ and $G_4 = (\!|Q_2|\!)_{\sigma_1}$ in Example 4.13. They fuse into: $\mathsf{q} \to \mathsf{p}{:}\{l_1{:}\mathsf{p} \to \mathsf{r}{:}\{l_2{:}\mathsf{end}\}\}$ or $\mathsf{q} \to \mathsf{p}{:}\{l_3{:}\mathsf{p} \to \mathsf{r}{:}\{l_4{:}\mathsf{end}\}\}$. Notably, the global type produced by our example cannot be captured in the work of [11, 14], requiring the two selections in branches l_1 and l_3 to be the same.

We make precise the claims of § 1, that is the network interconnections of global types induced by CLL are strictly less expressive than those of MP, by formalising the notion of interconnection network as an undirected graph.

Definition 5.4 (Interconnection Network). Given a partial global type G we generate its *Interconnection Network Graph* (ING) where the nodes are the roles of G and two nodes p, q share an edge iff $\mathsf{p} \leftrightarrow \mathsf{q}$ in G. ◇

We establish the main properties of our framework. The following proposition states that we can always fuse the partial global types of a well-formed composition.

Proposition 5.5. *Let $P \Vdash^\sigma_\rho \Delta; \Gamma; \mathcal{G}$ and $\Delta = \emptyset$ or Δ contains only $\mathbf{1}$ or $\perp$. There exists a single well-formed global type G such that $G = \mathsf{fuse}(\mathcal{G})$ where $\mathsf{fuse}(\mathcal{G})$ denotes fusion of all partial global types in $\mathcal{G}$.*

Proof. Since Δ is empty or contains only $\mathbf{1}$ or $\perp$ we have that P is an n-ary composition of (cut-free) processes. If $n = 1$ then $P = \mathbf{0}$ and its corresponding global type is just end. The interesting case is when $n > 1$.

Since the context is either empty or contains only $\mathbf{1}$ or $\perp$, we have that P is of the form $(\boldsymbol{\nu}\tilde{a})(P_1 \mid \cdots \mid P_n)$ where all free names $a_j{:}A_j$ of each of the P_i processes are cut with some other $P_{i'}$ using $a_j{:}A_j^\perp$. Thus, by construction of $\Vdash$ we have that for each bound name a of P we have $\mathsf{c}_\rho(a)[\mathsf{p}_\rho(a)]{:}T \in \Gamma$ and $\mathsf{c}_\rho(a)[\mathsf{d}_\rho(a)]{:}T' \in \Gamma$ with $T \upharpoonright \mathsf{d}_\rho(a) \leq \overline{T' \upharpoonright \mathsf{p}_\rho(a)}$ and thus for each action between two roles in a partial global type in $\mathcal{G}$, we can always find a matching action in another partial global type in $\mathcal{G}$, therefore we can fuse all partial global types in $\mathcal{G}$ into a single global type. □

The following main theorem shows that any two such partial types overlap in at most 2 roles; from which the acyclicity of CLL processes follows, establishing a separation between MP global types and those induced by CLL. Notice that Theorem 3.9 ensures that, in general, there exists no type- and thread-preserving translation from CLL to a single MP session where CLL has the same or more interconnectability than MP.

Theorem 5.6. *Let $P \Vdash^\sigma_\rho \Delta; \mathcal{G}$. For any distinct $G_1, G_2 \in \mathcal{G}$ we have that $\mathsf{roles}(G_1) \cap \mathsf{roles}(G_2)$ contains at most 2 elements.*

Proof. We proceed by induction on the derivation $P \Vdash^\sigma_\rho \Delta; \mathcal{G}$, showing that each case preserves the specified invariant of at most 2 elements in the intersection of $\mathsf{roles}(G_1) \cap \mathsf{roles}(G_2)$, for any $G_1, G_2 \in \mathcal{G}$.

The only interesting case is when the last rule in the derivation is (comp-$\mathcal{G}$):

$$\text{(comp-}\mathcal{G}\text{)}\ \frac{P \vdash^{\sigma}_{\mathsf{CL}} \Delta, x{:}A \quad Q \Vdash^{\sigma'}_{\rho} \Delta', x{:}A^{\perp}; \mathcal{G} \quad (\star) \text{ in (comp)}}{(\boldsymbol{\nu} x)(P \mid Q) \Vdash^{(\sigma' \cup \sigma)\setminus\{x\}}_{\rho'} \Delta, \Delta'; \mathcal{G} \cup \{(\!|P|\!)_{\sigma}\}}$$

By the inductive hypothesis. we have that for any $G'_1, G'_2 \in \mathcal{G}$, $\mathsf{roles}(G'_1) \cap \mathsf{roles}(G'_2)$ contains at most 2 elements.

By construction we know that roles in $\mathcal{G}$ must either appear in σ' (corresponding to role assignments to channels in Δ' and a) or ρ (corresponding to role assignments to bound names).

By inversion we know that $\forall z \in \Delta, y \in \Delta'.\mathsf{d}_{\sigma}(z) \neq \mathsf{d}_{\sigma'}(y)$, thus there are no common d_{σ} role assignments between σ and σ' to free names of the two processes beyond those for x. We also know that $\mathsf{p}_{\sigma}(x) = \mathsf{d}_{\sigma'}(x)$ and $\mathsf{d}_{\sigma}(x) = \mathsf{p}_{\sigma'}(x)$.

By the definition of $(\!|P|\!)_{\sigma}$ there are at least two common role names with each endpoint interaction in $\mathcal{G}$ coming from σ and σ' (i.e. role assignments to free names), which are p_{σ} and $\mathsf{p}_{\sigma'}$. Since p_{σ} is invariant and $\forall z \in \Delta, y \in \Delta'.\mathsf{d}_{\sigma}(z) \neq \mathsf{d}_{\sigma'}(y)$, we have that free names in Δ cannot share any additional roles.

We need now only consider ρ. By construction, we know that $\forall z \in \Delta, y \in \Delta'.\mathsf{d}_{\sigma}(z) \notin \rho \wedge \mathsf{d}_{\sigma'}(y) \notin \rho$, thus $\mathsf{d}_{\sigma}(z)$ cannot appear in $\mathcal{G}$ due to ρ. The only remaining possibilities are $\mathsf{p}_{\sigma}(x)$ and $\mathsf{d}_{\sigma}(x)$ which are already accounted for from the argument above. Thus we preserve the invariant and conclude the proof. □

We can then establish a main result of our work: The connection network graphs generated by CLL-typable processes are acyclic.

Theorem 5.7. *Let $P \Vdash^{\sigma}_{\rho} \Delta; \mathcal{G}$. Let $G = \mathsf{fuse}(\mathcal{G})$. The interconnection network graph for G is acyclic.*

Proof. Each endpoint interaction sequence in $\mathcal{G}$ denotes the contribution of a single endpoint role in the global conversation. By Theorem 5.6 we have that any distinct pair of partial global types in $\mathcal{G}$ shares at most 2 role names. This means that for any distinct roles $\mathsf{p}, \mathsf{q}, \mathsf{r}$, if $\mathsf{p} \nleftrightarrow \mathsf{q} \in G$ and $\mathsf{p} \nleftrightarrow \mathsf{r} \in G$ then neither $\mathsf{q} \to \mathsf{r}$ nor $\mathsf{r} \to \mathsf{q}$ or $\mathsf{q} \rightsquigarrow \mathsf{r}$ nor $\mathsf{r} \rightsquigarrow \mathsf{q}$ in G. Hence, in the connection graph of G we know that we cannot have triangles of the form $(\mathsf{p}, \mathsf{q}), (\mathsf{p}, \mathsf{r}), (\mathsf{r}, \mathsf{q})$ as edges.

We can then see that no cycles can be formed through a "diamond" – a sequence of edges of the form $(\mathsf{p}, \mathsf{q}), (\mathsf{p}, \mathsf{r}), (\mathsf{q}, t_1), \ldots, (\mathsf{t}_n, \mathsf{s}), (\mathsf{r}, \mathsf{v}_1), \ldots, (\mathsf{v}_m, \mathsf{s})$ – in the graph by the fact that at each composition step, processes can only share one free name (the one that is the focus of the composition rule since $\Delta \cup \Delta' = \emptyset$) and role assignments ($\forall z \in \Delta, y \in \Delta'.\mathsf{d}_{\sigma}(z) \neq \mathsf{d}_{\sigma'}(y)$), similarly for ρ, by $(\star)$ (c) in rule (comp) and Definition 3.1). If we could form a "diamond" cycle in the graph, then we have to be able to eventually compose processes sharing more than one name or with different roles mapping to the same channel name in order to connect both $(\mathsf{v}_m, \mathsf{s})$ and $(\mathsf{t}_n, \mathsf{s})$. That is, after composing v_m we cannot compose the implementation of t_n (or vice-versa), since it would violate the role assignment restriction of composition – $(\star)$ (c) in (comp) – which disallows identical destination roles. Moreover, in order to compose t_n and s it would have to be the case that s shares a channel with both t_n and v_m (which is untypable) or the process composition of implementations of v_m and t_n would have to map both roles to the same channel shared with the implementation of s – itself also a contradiction. □

Deadlock-freedom in MP**.** Theorem 5.10 states that our encoding produces a single multiparty session, that is, the fusing of all partial global types in a complete session is deadlock-free. To prove the theorem, we require the following lemmas.

Lemma 5.8. *Let $P \Vdash^{\sigma}_{\rho} \Delta; \Gamma; \mathcal{G}$. $\mathsf{co}(\Gamma)$ implies $\Delta = \emptyset$ or Δ contains only $\mathbf{1}$ or $\perp$ and $\sigma = \emptyset$.*

Proof. See Appendix A.3.1. □

Lemma 5.9. *Let* $P \Vdash^{\emptyset}_{\rho} \Delta; \Gamma; \mathcal{G}$ *with* $\Delta = \emptyset$ *or* Δ *containing only* $\mathbf{1}$ *or* $\bot$*. We have that* $\mathrm{co}(\Gamma)$.

Proof. See Appendix A.3.2. □

Theorem 5.10. *Let* $P \Vdash^{\emptyset}_{\rho} \Delta; \mathcal{G}$ *and* $\Delta = \emptyset$ *or* Δ *contains only* $\mathbf{1}$ *or* $\bot$*. Then we have:* (1) $P \to^* \mathbf{0}$*; and* (2) $\mathrm{fuse}(\mathcal{G})$ *is well-formed.*

Proof. By Propositions 2.3 and 5.5 and Theorem 4.9, together with Lemmas 5.8 and 5.9. □

From Proposition 3.6 and Theorem 5.10, it follows that:

Corollary 5.11. *If* $P \Vdash^{\emptyset}_{\rho} \Delta; \Gamma$ *and* $\Delta = \emptyset$ *or* Δ *contains only* $\mathbf{1}$ *or* $\bot$*, then* $\rho(P) \vdash_{\mathsf{MP}} \Gamma$ *and* $\rho(P)$ *is deadlock-free.*

Recall that Proposition 2.7 states that the MP typing discipline does not guarantee deadlock-freedom. Theorem 5.10 shows that the translation from CLL automatically identifies a set of deadlock-free MP processes.

6 HIGHER-ORDER CHANNEL PASSING

In this section we lift the restrictions put in place in § 3 on the $\otimes$ and $⅋$ connectives, enabling CLL processes to perform full higher-order channel passing which can be mapped to MP processes with delegation. We follow the lines of § 3 and § 5, extending the framework and earlier results to this more general setting, emphasising crucial differences.

Channel mappings. Full higher-order channel passing creates interleaved *multiple* sessions and instantiations of channels into input bound variables. For these reasons, we revise our mapping of § 3, allowing for processes that send channels to hold multiple roles in the same multiparty session. We also account for bound names, where (bound) CLL channels are mapped to distinct MP session channels.

We present our mapping with two definitions: a mapping for cut-free processes (Definition 6.1) and a well-formedness condition for delegation (Definition 6.2). In the former, the mapping is identical to that of § 3 but the MP channel identifier need not be unique among all channels. We also need to treat consistency of bound names. In the latter, we enforce that when the typing rule for delegation (i.e. the $\otimes$ rule) is applied, channels used by the subprocesses must implement a different principal role.

Definition 6.1 (Channel Mapping). Let $P \vdash_{\mathsf{CL}} \Delta$ without using the cut rule. We define a channel to role-indexed channel mapping of P as a pair of mappings (σ, η) such that: **(1)** for all distinct $x, y \in fn(P)$, $\sigma(x) = s[\mathrm{p}][\mathrm{q}]$ and $\sigma(y) = s'[\mathrm{p}'][\mathrm{q}']$ where if $s = s'$ then $\mathrm{p} = \mathrm{p}'$ and $\mathrm{q} \neq \mathrm{q}'$; **(2)** for all distinct $x, y \in bn(P)$, $\eta(x) = s[\mathrm{p}][\mathrm{q}]$ and $\eta(y) = s'[\mathrm{p}'][\mathrm{q}']$ where $s \neq s'$ and $s, s' \notin \sigma$; and **(3)** for all distinct $x, y \in bv(P)$, $\eta(x) = x[\mathrm{p}]$ and $\eta(y) = y[\mathrm{p}']$.

The restrictions in Definition 6.1 allow for different CLL channels to be mapped to different MP session channels. However, within a given MP session we enforce that the principal role implemented by the cut-free process must be the same (which identifies a single participant with a cut-free process). We also ensure that different channels mapped to the same MP session do not have the same destination role as in the previous mapping (Definition 3.1).

Crucially, the η component of Definition 6.1 tracks all instances of channel output **(2)** and input **(3)**, where sent channels are mapped to fresh (binary) MP session channels for which one of the endpoints is delegated. Dually, received channels are mapped to variables with a role assignment.

$$\varphi_P(\overline{x}\langle y\rangle.(P_1 \mid P_2)) \triangleq \begin{cases} (\boldsymbol{\nu}\mathrm{c}_\eta(y))\mathrm{c}_\sigma(x)[\mathrm{p}_\sigma(x)][\mathrm{d}_\sigma(x)]\langle \mathrm{c}_\eta(y)[\mathrm{d}_\eta(y)]\rangle.(\varphi_P(P_1) \mid \varphi_P(Q_2)) & \text{if } x \in \mathit{fn}(P) \\ (\boldsymbol{\nu}\mathrm{c}_\eta(y))x[\mathrm{d}_\eta(x)]\langle \mathrm{c}_\eta(y)[\mathrm{d}_\eta(y)]\rangle.(\varphi(P_1) \mid \varphi(Q_2)) & \text{if } x \in \mathit{bv}(P) \end{cases}$$

$$\varphi_P(x(y).Q) \triangleq \begin{cases} \mathrm{c}_\sigma(x)[\mathrm{p}_\sigma(x)][\mathrm{d}_\sigma(x)](y).\varphi_P(Q) & \text{if } x \in \mathit{fn}(P) \\ x[\mathrm{d}_\eta(x)](y).\varphi_P(Q) & \text{if } x \in \mathit{bv}(P) \end{cases}$$

Fig. 6. Process Mapping

Definition 6.2 (Delegation). Let $P \vdash_{\mathsf{CL}} \Delta$ without using the cut rule, and (σ, η) be a mapping viz. Definition 6.1. We say that (σ, η) is *well-formed* if the number of distinct MP channels in the image of σ is minimal and for each use of $\otimes$ in typing P we have:

$$\frac{P_1 \vdash_{\mathsf{CL}} \Delta_1, y{:}A \quad P_2 \vdash_{\mathsf{CL}} \Delta_2, x{:}B}{\overline{x}\langle y\rangle.(P_1 \mid P_2) \vdash_{\mathsf{CL}} \Delta_1, \Delta_2, x{:}A \otimes B}$$

(Role Disjointness) $\forall z \in \Delta_1, z' \in \Delta_2, x$: $\mathrm{c}_\sigma(z)=\mathrm{c}_\sigma(z')$ implies $\mathrm{p}_\sigma(z) \neq \mathrm{p}_\sigma(z')$;
(Role/Destination Disjointness) $\forall z \in \Delta_1, z' \in \Delta$, $z \neq z'$ and $\mathrm{c}_\sigma(z)=\mathrm{c}_\sigma(z')$ imply $\mathrm{d}_\sigma(z) \neq \mathrm{p}_\sigma(z')$.

$P \vdash^{(\sigma,\eta)}_{\mathsf{CL}} \Delta$ denotes (σ, η) is well-formed for delegation wrt $P \vdash_{\mathsf{CL}} \Delta$.

The conditions of Definition 6.2 ensure that channels that are used within parallel compositions (due to delegation in CLL) are assigned different principal roles within the same MP session (the condition **(Role Disjointness)**). This ensures typability in MP. Moreover, the **(Role/Destination Disjointness)** condition forbids all principal roles used in the process from being used as destination roles within the same process (i.e. disallowing two endpoints of a communication within the same thread).

Now we define a main mapping from CLL to MP.

Definition 6.3 (Process Mapping). Given $P \vdash_{\mathsf{CL}} \Delta$ without using the cut rule and a mapping (σ, η) according to Definition 6.1, we define the mapping from P to the MP process $\varphi_P(P)$, where $\varphi = \sigma \circ \eta$ according to the rules of Fig. 6. We often omit the subscript P when clear from context.

The following example illustrates how our mapping for higher-order channel passing transforms CLL into MP processes and the conditions imposed on the mapping.

Example 6.4. Let $P \triangleq \overline{x}\langle y\rangle.(P_1 \mid P_2)$ with $P_1 \triangleq \overline{w_1}\langle 1\rangle \mid \overline{y}\langle 2\rangle$ and $P_2 \triangleq \overline{w_2}\langle 3\rangle \mid \overline{x}\langle 5\rangle$. Then assume (we write φ for $\sigma \circ \eta$):

$$\varphi(P) = \quad (\boldsymbol{\nu}s')s[\mathrm{p}][\mathrm{q}]\langle s'[\mathrm{q}']\rangle.(s_1[\mathrm{p}_1][\mathrm{q}_1]\langle 1\rangle \mid s'[\mathrm{r}][\mathrm{q}']\langle 2\rangle \mid s_2[\mathrm{p}_2][\mathrm{q}_2]\langle 3\rangle \mid s[\mathrm{p}][\mathrm{q}]\langle 5\rangle)$$

for some well-formed η and σ. Then $s_2 \neq s$ and by Definition 6.2:

- if $s = s_1$, $\mathrm{p} \neq \mathrm{p}_1$ (by **(1)**) and $\mathrm{q} \neq \mathrm{p}_1$ and $\mathrm{p} \neq \mathrm{q}_1$ (by **(2)**); and
- if $s_1 = s_2$, $\mathrm{p}_1 \neq \mathrm{p}_2$ (by **(1)**) and $\mathrm{q}_1 \neq \mathrm{p}_2$ and $\mathrm{q}_2 \neq \mathrm{p}_1$ (by **(2)**).

Note that a valid mapping does not exclude the cases where two destination roles can be same, i.e. $\mathrm{q} = \mathrm{q}_1$ and $\mathrm{q}_1 = \mathrm{q}_2$ are allowed when $s = s_1$ and $s_1 = s_2$. However, composition with such mappings will be subsequently excluded by the parallel composition rule in Fig. 7.

Local type generation. Since cut-free processes may denote multiple roles, we generate a local typing *context* from each process, assigning local types to each role indexed channel. To generate the local type for a session output of the form $\overline{x}\langle y\rangle.(P \mid Q)$, where y occurs free in P but not in Q (and symmetrically for x), we produce the delegated session type from the usage of y in P. The two continuation processes are then inspected inductively. Since local type generation produces a local typing context instead of a single local type, we provide a combination operation $s[\mathrm{p}]{:}T \diamond \Gamma$ that acts as a simple union, but for type assignments for $s[\mathrm{p}] : T'$ in Γ results in an assignment in which T precedes T'. Note that, since P and Q use disjoint sets of channels (which are mapped to disjoint roles), applying $\diamond$ to the corresponding generated local types amounts to a simple set union.

The operation $s[\mathrm{p}]{:}T \diamond \Gamma$, that essentially appends a local type assignment $s[\mathrm{p}]{:}T$ to those in Γ that match the session channel and role, is defined as follows (recall $\dagger$ denotes either $\{\uparrow, \downarrow, \oplus, \&\}$):

$$\begin{array}{lcll}
s[\mathrm{p}]{:}\mathrm{q}\dagger(T) \diamond (\Gamma', s[\mathrm{p}]{:}T') & \triangleq & \Gamma', s[\mathrm{p}]{:}\mathrm{q}\dagger(T); T' & \\
s[\mathrm{p}]{:}\mathrm{q}\dagger(T) \diamond \Gamma' & \triangleq & s[\mathrm{p}]{:}\mathrm{q}\dagger(T); \mathsf{end}, \Gamma' & \text{with } s[\mathrm{p}] \notin \Gamma' \\
s[\mathrm{p}]{:} \dagger\, \mathrm{q}\{l_j : T_j\}_{j\in J} \diamond (\Gamma', s[\mathrm{p}]{:}T') & \triangleq & \Gamma', s[\mathrm{p}]{:} \dagger\, \mathrm{q}\{l_j : T_j\}_{j\in J} & \\
s[\mathrm{p}]{:} \dagger\, \mathrm{q}\{l_j : T_j\} \diamond \Gamma' & \triangleq & s[\mathrm{p}]{:} \dagger\, \mathrm{q}\{l_j : T_j\}, \Gamma' & \text{with } s[\mathrm{p}] \notin \Gamma'
\end{array}$$

Intuitively, $s[\mathrm{p}]{:}T \diamond \Gamma$ is $\Gamma, s[\mathrm{p}]{:}T$ if $s[\mathrm{p}] \notin \Gamma$. Otherwise, we have that $\Gamma = \Gamma', s[\mathrm{p}] : T'$ and thus we modify T' to $T; T'$ if T is an input our output; or, change T' to T if T is a selection or branching (since T will already contain all actions of role p by construction of the local type generation procedure).

Definition 6.5 (Local Type Generation). Let $P \vdash_{\mathsf{CL}}^{(\sigma,\eta)} \Delta$ where all free and bound names are distinct. We generate a local typing context Γ such that $\eta(\sigma(P)) \vdash_{\mathsf{MP}} \Gamma$ by induction on the structure of P, written $[\![P]\!]^\eta_\sigma$. Below we write $\mathrm{c}_\varphi(x)$, $\mathrm{p}_\varphi(x)$ and $\mathrm{d}_\varphi(x)$, where φ stands for σ if $x \in \mathit{fn}(P)$ and η otherwise; and c for $\mathrm{c}_\varphi(x)[\mathrm{p}_\varphi(x)]$ if $x \in \mathit{fn}(P)$ and for x otherwise; q for $\mathrm{d}_\varphi(x)$; $[\![P]\!]^\eta_\sigma(y)$ denotes the binding for y in the generated context):

$$\begin{array}{lcl}
[\![\mathbf{0}]\!]^\eta_\sigma & \triangleq & \emptyset \\
[\![\overline{x}\langle y\rangle.(P \mid Q)]\!]^\eta_\sigma & \triangleq & c{:}\mathrm{q}\uparrow(\overline{[\![P]\!]^\eta_\sigma(y)}) \diamond ([\![P]\!]^\eta_\sigma \diamond [\![Q]\!]^\eta_\sigma) \\
[\![x(y).P]\!]^\eta_\sigma & \triangleq & c{:}\mathrm{q}\downarrow([\![P]\!]^\eta_\sigma(y)) \diamond ([\![P]\!]^\eta_\sigma \setminus \eta(y)) \\
[\![x.l_j; P]\!]^\eta_\sigma & \triangleq & c{:} \oplus \mathrm{q}\{l_j{:}[\![P]\!]^\eta_\sigma(c)\} \diamond [\![P]\!]^\eta_\sigma \\
[\![x.\mathsf{case}\{l_i : P_i\}_{i\in I}]\!]^\eta_\sigma & \triangleq & c{:} \&\, \mathrm{q}\{l_i{:}[\![P_i]\!]^\eta_\sigma(c)\}_{i\in I} \diamond [\![P]\!]^\eta_\sigma
\end{array}$$

where $\overline{T}$ denotes a dual type of T in a session with two roles p, q as follows:

$$\overline{\mathrm{p}\uparrow(T); S} \triangleq \mathrm{q}\downarrow(T); \overline{S} \qquad \overline{\mathrm{p}\downarrow(T); S} \triangleq \mathrm{q}\uparrow(T); \overline{S}$$
$$\overline{\&\mathrm{p}\{l_i : T_i\}_{i\in I}} \triangleq \oplus\mathrm{q}\{l_i : \overline{T_i}\}_{i\in I} \quad \overline{\oplus\mathrm{p}\{l_i : T_i\}_{i\in I}} \triangleq \&\mathrm{q}\{l_i : \overline{T_i}\}_{i\in I} \quad \overline{\mathsf{end}} \triangleq \mathsf{end}$$

We note that the carried type in outputs is dualised in order to match with session type duality.

Example 6.6. Consider:

$$P \triangleq \overline{x}\langle y\rangle.(y(n).y(m).z\langle n\rangle.\mathbf{0} \mid \mathbf{0}) \qquad Q \triangleq x(y).y\langle 0\rangle.y\langle 1\rangle.\mathbf{0}$$

with the following typings:

$$P \vdash_{\mathsf{CL}} x{:}(\mathsf{nat} \parr \mathsf{nat} \parr \bot) \otimes \mathbf{1}, z{:}\mathsf{nat} \otimes \mathbf{1} \qquad Q \vdash_{\mathsf{CL}} x{:}(\mathsf{nat} \otimes \mathsf{nat} \otimes \mathbf{1}) \parr \bot$$

We can produce mappings σ_1, η_1 and σ_2, η_2 such that (we write φ_i for $\sigma_i \circ \eta_i$):

$$\begin{array}{lcl}
\varphi_1(P) & = & (\boldsymbol{\nu} s')s[\mathrm{p}][\mathrm{q}]\langle s'[\mathrm{q}']\rangle; (s'[\mathrm{p}'][\mathrm{q}'](n); s'[\mathrm{p}'][\mathrm{q}'](m); s[\mathrm{t}][\mathrm{v}]\langle n\rangle; \mathbf{0} \mid \mathbf{0}) \\
\varphi_2(Q) & = & s[\mathrm{q}][\mathrm{p}](y); y[\mathrm{p}']\langle 0\rangle; y[\mathrm{p}']\langle 1\rangle; \mathbf{0}
\end{array}$$

and we have that:

$$(\nu x)(P \mid Q) \vDash^{\sigma_3,\eta} z{:}\mathsf{nat} \otimes \mathbf{1}; s[\mathrm{p}]{:}T_1, s[\mathrm{q}]{:}T_2, s[\mathrm{t}]{:}T_3$$

$$(\mathsf{comp}_d)\dfrac{P \vdash^{\sigma,\eta}_{\mathsf{CL}} \Delta, x{:}A \qquad Q \vDash^{\sigma',\eta'}_{\rho} \Delta', x{:}A^{\perp}; \Gamma; \mathcal{G} \quad (\dagger)}{(\boldsymbol{\nu} x)(P \mid Q) \vDash^{(\sigma' \cup \sigma)\setminus\{x\}, \eta\cup\eta'}_{\rho'} \Delta, \Delta'; \Gamma, [\![P]\!]^{\eta}_{\sigma}; \mathcal{G} \cup (\!|P|\!)^{\eta}_{\sigma}}$$

(†) **(a)** bound channel: $\rho' = \rho \cup (x, \mathsf{c}_\sigma(x)[\mathsf{p}_\sigma(x)][\mathsf{d}_\sigma(x)])$
(b) role/destination match: $\mathsf{c}_\sigma(x) = \mathsf{c}_{\sigma'}(x) \wedge \mathsf{p}_\sigma(x) = \mathsf{d}_{\sigma'}(x) \wedge \mathsf{d}_\sigma(x) = \mathsf{p}_{\sigma'}(x)$
$\forall z \in \Delta, y \in \Delta'. \mathsf{c}_\sigma(z) = \mathsf{c}_{\sigma'}(y) \Rightarrow$
(c) unique destination: $(\mathsf{d}_\sigma(z) \neq \mathsf{d}_{\sigma'}(y) \wedge \mathsf{d}_\sigma(z), \mathsf{d}_{\sigma'}(y) \notin \rho \wedge$
(d) role/destination disjointness: $\mathsf{p}_\sigma(z) \neq \mathsf{d}_{\sigma'}(y) \wedge \mathsf{d}_\sigma(z) \neq \mathsf{p}_{\sigma'}(y) \wedge \mathsf{p}_\sigma(z) \neq \mathsf{p}_{\sigma'}(y))$
(e) bound role/destination match: $\forall x \in \eta. \forall y \in \eta'. x = y \Rightarrow \mathsf{d}_\eta(x) = \mathsf{p}_{\eta'}(y) \vee \mathsf{p}_\eta(x) = \mathsf{d}_{\eta'}(y)$
(f) bound channel disjointness: $\forall x \in \eta. \forall y \in \varphi = \{\sigma', \eta', \rho\}. \mathsf{c}_\eta(x) \neq \mathsf{c}_\varphi(y)$

Fig. 7. Parallel Composition Mapping with Channel Passing

with

$$\begin{array}{rcl} T & = & \mathsf{p}'{\uparrow}(\mathsf{nat}); \mathsf{p}'{\uparrow}(\mathsf{nat}); \mathsf{end} \\ T_1 & = & \mathsf{q}{\uparrow}(T); \mathsf{end} \\ T_2 & = & \mathsf{p}{\downarrow}(\mathsf{p}'{\uparrow}(\mathsf{nat}); \mathsf{p}'{\uparrow}(\mathsf{nat}); \mathsf{end}); \mathsf{end} \\ T_3 & = & \mathsf{v}{\uparrow}(\mathsf{nat}); \mathsf{end} \end{array}$$

Note that mapping z to $s[\mathsf{p}][\mathsf{t}]$, for instance, would not allow for a valid composition of the two processes since we would have the role p of session s spread across two threads. Likewise, mapping z to $s[\mathsf{t}][\mathsf{q}]$ would disallow the composition of P and Q since it would require the two processes to share two distinct channel names.

Composition and interconnection networks. We define composition in tandem with the partial global types for delegation as other constructs are identical to Definition 5.1 (with a single session). We introduce the judgement $P \vDash^{\sigma,\eta}_{\sigma} \Delta; \Gamma; \mathcal{G}$, where (σ, η) is a well-formed mapping according to Definition 6.2, following a similar pattern to the composition judgement of § 3 and § 5.

Definition 6.7 (Partial Global Types). Given $P \vdash^{(\sigma,\eta)}_{\mathsf{CL}} \Delta$ we generate its partial global type wrt s, written $(\!|P|\!)^{\eta}_{\sigma}(s)$ by induction on the structure of P (♯ denotes $x \in \sigma \wedge \mathsf{c}_\sigma(x) = s$):

$$(\!|x\langle y\rangle.(Q_1 \mid Q_2)|\!)^{\eta}_{\sigma}(s) \triangleq \begin{cases} \mathsf{p}_\sigma(x) \rightsquigarrow \mathsf{d}_\sigma(x) :\uparrow (\overline{[\![Q_1]\!]^{\eta}_{\sigma}(y)}).\mathsf{fuse}((\!|Q_1|\!)^{\eta}_{\sigma}(s), (\!|Q_2|\!)^{\eta}_{\sigma}(s)) & (\sharp) \\ \mathsf{fuse}((\!|Q_1|\!)^{\eta}_{\sigma}(s), (\!|Q_2|\!)^{\eta}_{\sigma}(s)) & \text{otherwise} \end{cases}$$
$$(\!|x(y).Q|\!)^{\eta}_{\sigma}(s) \triangleq \begin{cases} \mathsf{d}_\sigma(x) \rightsquigarrow \mathsf{p}_\sigma(x) :\downarrow ([\![Q]\!]^{\eta}_{\sigma}(y)).(\!|Q|\!)^{\eta}_{\sigma}(s) & (\sharp) \\ (\!|Q|\!)^{\eta}_{\sigma}(s) & \text{otherwise} \end{cases}$$

Let C be the set of session channels in the image of σ. We denote by $(\!|P|\!)^{\eta}_{\sigma}$ the set $\bigcup((\!|P|\!)^{\eta}_{\sigma}(s))_{s\in C}$.

We define $P \vDash^{\sigma,\eta}_{\sigma} \Delta; \Gamma; \mathcal{G}$, where (σ, η) is a well-formed mapping, with the rule (changes wrt (★) appear in red) in Fig. 7. Following the Barendregt convention, we assume that if $x \in bn(P) \cap bv(Q)$ or $x \in bv(P) \cap bn(Q)$, then P and Q eventually exchange x. Conditions **(a)**-**(c)** are identical to (comp) in Fig. 4, but where we refer explicitly to the mapping of CLL channels to MP session channels. Condition **(d)** ensures that P and Q are only connected via the channel x, as required by CLL, and that no roles are split across the two processes. Conditions **(e)**-**(f)** ensure that channels that are to be exchanged are mapped to fresh channels and with consistent role assignments in MP.

Example 6.8. The following processes are untypable in the global progress type system of [18].

$$\begin{array}{lcl} P & \triangleq & x\langle y\rangle.y(n).\mathbf{0} \vdash_{\mathsf{CL}} x{:}(\mathsf{int} \parr \bot) \otimes \mathbf{1} \\ Q & \triangleq & z\langle w\rangle.w\langle 1\rangle.\mathbf{0} \vdash_{\mathsf{CL}} z{:}(\mathsf{int} \otimes \mathbf{1}) \otimes \mathbf{1} \\ R & \triangleq & x(y).z(w).w(n).y\langle n\rangle.\mathbf{0} \vdash_{\mathsf{CL}} x{:}(\mathsf{int} \otimes \mathbf{1}) \parr \bot, z{:}(\mathsf{int} \parr \bot) \parr \bot \end{array}$$

Given mappings $\sigma_1, \sigma_2, \sigma_3, \eta_1, \eta_2, \eta_3$ we obtain the following (with $\varphi_i = \sigma_i \circ \eta_i$):

$$\begin{array}{l} \varphi_1(P) = (\boldsymbol{\nu} t)s[\mathsf{p}][\mathsf{r}]\langle t[\mathsf{t}]\rangle; t[\mathsf{s}][\mathsf{t}](n); \mathbf{0} \\ \varphi_2(Q) = (\boldsymbol{\nu} t')s'[\mathsf{q}][\mathsf{b}]\langle t'[\mathsf{s}]\rangle; t'[\mathsf{r}][\mathsf{s}]\langle 1\rangle; \mathbf{0} \\ \varphi_3(R) = s[\mathsf{r}][\mathsf{p}](y); s'[\mathsf{b}][\mathsf{q}](w); w[\mathsf{s}](n); y[\mathsf{t}]\langle n\rangle; \mathbf{0} \end{array}$$

$$[\![P]\!]^{\sigma_1}_{\eta_1} = s[\mathsf{p}]{:}\mathsf{r}\uparrow(\mathsf{s}\uparrow(\mathsf{int})) \quad [\![Q]\!]^{\sigma_2}_{\eta_2} = s'[\mathsf{q}]{:}\mathsf{b}\uparrow(\mathsf{r}\downarrow(\mathsf{int})) \text{ and } [\![R]\!]^{\sigma_3}_{\eta_3} = s[\mathsf{r}]{:}\mathsf{p}\downarrow(\mathsf{s}\uparrow(\mathsf{int})), s'[\mathsf{b}]{:}\mathsf{q}\downarrow(\mathsf{r}\downarrow(\mathsf{int}))$$

We showcase a form of systems that cannot be directly represented in the MCP system of [11] as a single session.

Example 6.9 (Comparison with [11, 14]). Consider the following CLL processes, where P_1 employs channel passing to send a session of type int ⊗ **1** (we write φ_i for $\sigma_i \circ \eta_i$):

$$\begin{array}{l} P_1 \triangleq \overline{a}\langle y\rangle.(b(x).y\langle x\rangle \mid a\langle\rangle) \vdash_{\mathsf{CL}} a{:}(\mathsf{int} \otimes \mathbf{1}) \otimes \mathsf{unit} \otimes \mathbf{1}, b{:}\mathsf{int} \parr \bot \\ P_2 \triangleq a(y).y(x).a() \vdash_{\mathsf{CL}} a{:}(\mathsf{int} \parr \bot) \parr \mathsf{unit} \parr \bot, c{:}\mathbf{1} \\ P_3 \triangleq b\langle 33\rangle \vdash_{\mathsf{CL}} b{:}\mathsf{int} \otimes \mathbf{1} \end{array}$$

We can define mappings $\sigma_1, \sigma_2, \sigma_3, \eta_1, \eta_2, \eta_3$ such that:

$$\begin{array}{l} \varphi_1(P_1) = (\boldsymbol{\nu} s')s[\mathsf{p}][\mathsf{q}]\langle s'[\mathsf{q}_2]\rangle; (s[\mathsf{t}][\mathsf{r}](x); s'[\mathsf{p}_2][\mathsf{q}_2]\langle x\rangle \mid s[\mathsf{p}][\mathsf{q}]\langle\rangle) \\ \varphi_2(P_2) = s[\mathsf{q}][\mathsf{p}](y); y[\mathsf{p}_2](x); s[\mathsf{q}][\mathsf{p}]() \\ \varphi_3(P_3) = s[\mathsf{r}][\mathsf{t}]\langle 33\rangle \end{array}$$

We can compose the three processes using our (comp_d) rule such that the corresponding global type G is $\mathsf{p} \to \mathsf{q} : (T).\mathsf{r} \to \mathsf{t} : (\mathsf{int}).\mathsf{p} \to \mathsf{q} : (\mathsf{unit})$. Note that MCP of [11] cannot type this composition using G, since P_1 has actions of both p and t of s. To type a composition of this form in MCP, we have two options: (1) we force the actions on CLL channel b correspond to a separate MP session, thus requiring two global types ($G_1 = \mathsf{p} \to \mathsf{q} : (T).\mathsf{p} \to \mathsf{q} : (\mathsf{unit})$ and $G_2 = \mathsf{r} \to \mathsf{t} : (\mathsf{int})$) to type the corresponding processes; or (2), we separate role t into an independent fourth process $P_4 = s[\mathsf{t}][\mathsf{r}](x)$, removing the communication from P_1 and requiring an additional *thread* at the start of the session (note the dependency between the input from r and the output on s' is lost).

Consistency and acyclicity. Given that a cut-free process might contain many sessions and roles per session (i.e. multiple threads), we extend the notion of a thread and thread-preservation.

Informally we can regard a cut-free process P as a thread if (1) each sequential subterm of P contains only one role per session; and (2) in each delegation subterm of P of the form $\overline{x}\langle y\rangle.(Q \mid R)$, the delegated channel y is allocated to a different session. Condition (1) means that the mapping builds the longest (i.e. with the most communication steps) possible typable session, and condition (2) avoids self-delegation. This is consistent with the conditions of Definition 6.1 and Definition 6.2.

In order to state a similar property to Theorem 3.9, we move from a single session to multiple sessions, where bijective renaming φ ensures that (1) if different channels map to the same session, their destination roles differ; and, (2) if the same session channel appears in two parallel prefixes, their principal and destination roles are pairwise distinct. Since judgement $P \vDash^{\sigma,\eta}_{\rho} \Delta; \Gamma; \mathcal{G}$ relies on the channel mappings in Definitions 6.1 and 6.2, we prove the following property which corresponds to Theorem 3.9 in this channel-passing setting, under the assumption that ρ, σ and η conform the channel mappings.

Theorem 6.10 (Consistency). *Assume $P \vdash_{\mathsf{CL}} \Delta$. If $\varphi(P)$ is typable by Γ, i.e. $\varphi(P) \vdash_{\mathsf{MP}} \Gamma$ and ρ, σ and η satisfy the conditions in Definitions 6.1 and 6.2 and $\varphi = \sigma \circ \eta \circ \rho$, then $P \models^{\sigma,\eta}_{\rho} \Delta; \Gamma; \mathcal{G}$.*

Proof. See Appendix A.4.1. □

The property above, while weaker than the uniqueness property of Theorem 3.9, provides a form of consistency between the channel mappings and the (comp_d) rule in Fig. 7.

Proposition 6.11. *Let $P \models^{\sigma,\eta}_{\rho} \Delta; \Gamma; \mathcal{G}$ and $\Delta = \emptyset$ or Δ contains only $\mathbf{1}$ or $\bot$. There exists a single well-formed global type G, such that $G = \mathsf{fuse}_s(\mathcal{G})$ where $\mathsf{fuse}_s(\mathcal{G})$ denotes fusion of all partial global types for session s in $\mathcal{G}$.*

Proof. Identical to Proposition 5.5 due to the fact that instances of session delegation that are composed in P are mapped to independent and complete global types. □

We prove that delegation does not add to the connection graph since delegation denotes a distinct multiparty session (i.e. the analogue of Theorem 6.12).

Theorem 6.12. *Let $P \models^{\sigma,\eta}_{\rho} \Delta; \Gamma; \mathcal{G}$ and $\Delta = \emptyset$ or Δ containing only $\mathbf{1}$ or $\bot$. The interconnection network graph for $\mathsf{fuse}_s(\mathcal{G})$ for each session s 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 either:

(1) *Triangle*: $(\mathsf{p}, \mathsf{q}), (\mathsf{p}, \mathsf{r}), (\mathsf{q}, \mathsf{r})$; or
(2) *Diamond*: $(\mathsf{p}, \mathsf{q}), (\mathsf{p}, \mathsf{r}), (\mathsf{q}, \mathsf{t}_1), \dots, (\mathsf{t}_n, \mathsf{s}), (\mathsf{r}, \mathsf{v}_1), \dots (\mathsf{v}_m, \mathsf{s})$.

We show that Δ cannot be empty or contain only $\mathbf{1}$ or $\bot$ in either case, deriving a contradiction. The proof follows the general lines of that of Theorem 5.7, but with a more involved case analysis since a cut-free process may now implement multiple roles (which by the restrictions of Definitions 6.1 and 6.2 will not be connected), and thus we need to account for all possibilities of roles being implemented by the same process. In particular, the case for "diamond" shaped connections (**Case (2)** below) of the form $(\mathsf{p}, \mathsf{q}), (\mathsf{p}, \mathsf{r}), (\mathsf{q}, \mathsf{t}_1), \dots, (\mathsf{t}_n, \mathsf{s}), (\mathsf{r}, \mathsf{v}_1), \dots (\mathsf{v}_m, \mathsf{s})$ requires us to also account for the fact that r and q might be implemented by the same cut-free process, and similarly for q and v_1, r and t_1, and so on.

Case (1): Assume the connection graph for G contains a triangle. 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 r. Thus, we must have at least three process threads (P_1, P_2 and P_3), one for each role. Without loss of generality, assume $P_1 \vdash^{\sigma_0}_{\mathsf{CL}} x{:}A, y{:}B$ with $\sigma_0(x) = s[\mathsf{p}][\mathsf{q}]$ and $\sigma(y) = s[\mathsf{p}][\mathsf{r}]$. $P_2 \vdash^{\sigma_1}_{\mathsf{CL}} x : A^{\bot}, z{:}C$ with $\sigma_1(x) = s[\mathsf{q}][\mathsf{p}]$ and $\sigma_1(z) = s[\mathsf{q}][\mathsf{r}]$. It is then immediate that we cannot find any P_3 implementing r (to fully empty the context) since it would have to share two channel names with the composition of P_1 and P_2, which is not a well-formed composition according to (†) in rule (comp_d).

Case (2): Assume the connection graph for G contains a diamond. We already know by Theorem 5.7 that when all roles are implemented by separate process threads that we cannot form a diamond. Hence, by Definitions 6.1 and 6.2, it must be the case that unconnected roles in the graph are implemented in the same process.

We note that if q and r are implemented by the same process thread, then we cannot find a closing instance of p (since we would need to compose two processes sharing two channel names, which is disallowed by rule (comp_d), or with ill-formed mappings according to (†) in the same rule).

We proceed by case analysis on (n, m).

Case (2-1) $n = 0$ **and** $m = 0$**:** we cannot find a closed instance of the network since either it is the case that each role is assigned to each thread (and so Theorem 5.7 applies) or p and s are implemented by the same thread. If this were the case, we must have $P_1 \vdash^{\sigma_0}_{\mathsf{CL}} x{:}A, y{:}B, z{:}C, w{:}D$ such that $\sigma_0(x) = s[\mathsf{p}][\mathsf{q}]$, $\sigma_0(y) = s[\mathsf{p}][\mathsf{r}]$, $\sigma_0(z) = s[\mathsf{s}][\mathsf{r}]$ and $\sigma_0(w) = s[\mathsf{s}][\mathsf{q}]$. This is not possible by Definitions 6.1 and 6.2.

Case (2-2) $n = 0$ **and** $m = m' + 1$**:** we have established that (p and q), (p and r) and (q and r) cannot be implemented by the same thread. If q and v_1 were implemented by the same thread, we cannot find a closed instance of this network since we need to compose with the (distinct) implementations of p and r which themselves are connected (by the assumption of the existence of a diamond connection). Thus we need to compose a process mapped to ($s[\mathsf{q}][\mathsf{p}]$, $s[\mathsf{v}_1][\mathsf{r}]$ and $s[\mathsf{v}_1][\mathsf{v}_2]$) with a process mapped to ($s[\mathsf{p}][\mathsf{q}]$ and $s[\mathsf{p}][\mathsf{r}]$) and another mapped to ($s[\mathsf{r}][\mathsf{p}]$ and $s[\mathsf{r}][\mathsf{v}_1]$). If we compose the first with the second, we cannot compose with the third since two channel/role assignment pairs ($s[\mathsf{r}][\mathsf{p}]$ and $s[\mathsf{r}][\mathsf{v}_1]$) are shared, which is forbidden by (†) in rule (comp_d). A similar reasoning applies to the other ways of composing the processes. The same reasoning applies if q and v_i were implemented by the same thread.

Case (2-3) $n = n'+1$ **and** $m = m'+1$**:** we begin with the case where q and v_1 are implemented by the same thread and r and t_1 are implemented by the same thread, which are obviously uncomposable according to rule (comp_d), as two distinct channels are shared. The same reasoning applies as we increase n and m. If t_i and v_i are implemented by the same thread we must have processes sharing two distinct channels, which is impossible. If t_n and v_m are mapped to 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, the same reasoning described in **Case (2-1)** case applies. □

Lemma 6.13. *Let* $P \vDash^{\emptyset,\eta}_{\rho} \Delta; \Gamma; \mathcal{G}$. $\mathsf{co}(\Gamma)$ *implies* $\Delta = \emptyset$ *or* Δ *containing only* $\mathbf{1}$ *or* $\bot$.

Proof. We note that the renamings ensure that the two endpoints of an interaction cannot be implemented by the same single-thread process and that bound-names involved in delegation denote linear interactions along different session channels. Hence a similar argument using the contradiction is applicable (with a case analysis for delegations) as the proof of Lemma 5.8. See Appendix A.4.2 for the details. □

Lemma 6.14. *Let* $P \vDash^{\emptyset,\eta}_{\rho} \Delta; \Gamma; \mathcal{G}$ *with* $\Delta = \emptyset$ *or* Δ *containing only* $\mathbf{1}$ *or* $\bot$*. We have that* $\mathsf{co}(\Gamma)$.

Proof. We account for the additional case of delegation by noting that the sent channel has a dual behaviour to the received channel (and we generate compatible endpoint types in Γ). The result then follows using a similar argument as the proof of Lemma 5.9. □

Theorem 6.15. *Let* $P \vDash^{\sigma,\eta}_{\rho} \Delta; \Gamma; \mathcal{G}$ *and* $\Delta = \emptyset$ *or* Δ *containing only* $\mathbf{1}$ *or* $\bot$*. Then we have:* (1) $P \rightarrow^* \mathbf{0}$*; and* (2) $\mathsf{fuse}_s(\mathcal{G})$ *at each session* s *is well-formed and deadlock-free.*

Proof. (1) follows from [9, 60] and (2) follows from Proposition 6.11. □

Corollary 6.16. *If* $P \vDash^{\emptyset,\eta}_{\rho} \Delta; \Gamma; \mathcal{G}$ *and* $\Delta = \emptyset$ *or* Δ *contains only* $\mathbf{1}$ *or* $\bot$*, then* $\eta(\rho(P))$ *is deadlock-free.*

7 REPLICATION

We account for CLL replication within our framework of MP *types*. Typically, MP processes (and types) do not explicitly account for replication, thus instead of extending the MP process calculus with a non-standard form of MP replication that corresponds to the binary sessions of CLL, we reason at the level of types. From the point of view of interconnectability *within* a linear session, the replication construct consists of the ability to "repeat" existing interconnections in *fresh*, independent sessions. While a replicated session may be used by *many* processes (due to the (cut$^!$) rule which allows for many threads to use a replicated name, akin to a multicut for non-linear sessions), each session replica is necessarily independent since rule (!) does not allow a replicated session to rely on other linear sessions. Hence the addition of replication does not change the nature of interconnectability of linear sessions in CLL – each client of a replicated session is connected to a distinct, isolated instance. Given that we focus on the interconnectability of CLL (hence generation of global and local types), we establish our main results by extending global and local *types* (but not the MP process syntax) with a form of replication.

Channel mapping. We define our mapping as in Definition 6.1, but we allow for different session names in the same thread and multiple destination roles for replicated channel names (otherwise the mapping of replicated inputs would be degenerate). Given $P \vdash_{\mathsf{CL}} \Xi; \Delta$ we enforce that the channels in Ξ and Δ be mapped to distinct MP session channels. Moreover, we require that all linear channels which deal with replicated sessions be mapped to distinct MP sessions. We write $\mathsf{posBang}(A)$ iff $!/?$ occurs to the right of $\otimes$ or $⅋$, or in a selection or branching in A. This predicate ensures that the behaviour specified by A aims to offer or use a replicated session but not delegate such a session. For the case of replicated inputs, p_σ denotes a single role and d_σ denotes a set of roles. The rest is unchanged.

Definition 7.1 (Channel Mapping – Replication). Let $P \vdash_{\mathsf{CL}} \Xi; \Delta$ without using the cut rule. We define a channel to role-indexed channel mapping of P as a pair of mappings (σ, η) such that: **(1)** for all distinct $x, y \in fn(P)$, then if $x \in \Xi$ and $y \in \Delta$ or $x, y \in \Xi$ then $\mathsf{c}_\sigma(x) \neq \mathsf{c}_\sigma(y)$. If $x, y \in \Delta$ then $\sigma(x) = s[\mathsf{p}][\mathsf{q}]$ and $\sigma(y) = s'[\mathsf{p}'][\mathsf{q}']$ where if $s = s'$ then $\mathsf{p} = \mathsf{p}'$ and $\mathsf{q} \neq \mathsf{q}'$; **(2)** for all distinct $x, y \in bn(P)$, $\eta(x) = s[\mathsf{p}][\mathsf{q}]$ and $\eta(y) = s'[\mathsf{p}'][\mathsf{q}']$ where $s \neq s'$ and $s, s' \notin \sigma$; **(3)** for all distinct $x, y \in bv(P)$, $\eta(x) = x[\mathsf{p}]$ and $\eta(y) = y[\mathsf{p}']$; and **(4)** $\forall x{:}A, y{:}B \in \Delta$ with $x \neq y$ if $\mathsf{posBang}(A)$ then $\mathsf{c}_\sigma(x) \neq \mathsf{c}_\sigma(y)$.

Clause **(1)** above, beyond the identical condition from Definition 6.1, ensures that replicated CLL session channels are mapped to distinct MP sessions that do not clash with those for linear CLL channels; clauses **(2)** and **(3)** deal with the treatment of bound channel names and variables, as in Definition 6.1; finally, clause **(4)** ensures that different CLL channels that aim to offer or use replicated session behaviours (i.e. whose types have occurrences of ! or ? that are not delegated) are not mapped to clashing MP session channels.

Global types and type generation. We extend the syntax of global types with the constructs for replication. Partial global types with a dedicated replication construct are $\tilde{\mathsf{p}} \rightsquigarrow \mathsf{q} : !(T).G$, with the corresponding dual $\mathsf{p} \rightsquigarrow \mathsf{q} : ?(T).G$ which fuse to the corresponding $\tilde{\mathsf{p}} \to \mathsf{q} : *(T).G$ global type, denoting that role q hosts the replicated behaviour T, to be used by roles $\tilde{\mathsf{p}}$ an arbitrary (but finite) number of times.

We then extend the partial global type generation to account for replication as follows.

Definition 7.2 (Type Generation). Let $P \vdash^{(\sigma,\eta)}_{\mathsf{CL}} \Xi; \Delta$, with all free and bound names distinct. We generate a set of role-indexed channels and types and partial global types wrt a multiparty session channel s, $\llbracket P \rrbracket^\eta_\sigma$ and $(\!| P |\!)^\eta_\sigma(s)$, respectively, by induction on the structure of P, as follows (we assume the same notations of Definition 6.5 and 6.7):

$$
\begin{array}{rcl}
[\![!x(y).P]\!]^{\eta}_{\sigma} & \triangleq & c{:}\mathsf{d}_{\varphi}(x)!([\![P]\!]^{\eta}_{\sigma}(y)) \uplus ([\![P]\!]^{\eta}_{\sigma} \setminus \eta(y)) \\
[\![\overline{x}\langle y\rangle.P]\!]^{\eta}_{\sigma} & \triangleq & c{:}\mathsf{d}_{\varphi}(x)?(\overline{[\![P]\!]^{\eta}_{\sigma}(y)}) \uplus ([\![P]\!]^{\eta}_{\sigma} \setminus \eta(y)) \\
(\!|!x(y).Q|\!)^{\eta}_{\sigma}(s) & \triangleq & \begin{cases} \mathsf{d}_{\sigma}(x) \rightsquigarrow \mathsf{p}_{\sigma}(x) : !([\![Q]\!]^{\eta}_{\sigma}(y)).(\!|Q|\!)^{\eta}_{\sigma}(s) & (x \in \sigma \wedge \mathsf{c}_{\sigma}(x) = s) \\ (\!|Q|\!)^{\eta}_{\sigma}(s) & \text{otherwise} \end{cases} \\
(\!|\overline{x}\langle y\rangle.Q|\!)^{\eta}_{\sigma}(s) & \triangleq & \begin{cases} \mathsf{p}_{\sigma}(x) \rightsquigarrow \mathsf{d}_{\sigma}(x) :?(\overline{[\![Q]\!]^{\eta}_{\sigma}(y)}).(\!|Q|\!)^{\eta}_{\sigma}(s) & (x \in \sigma \wedge \mathsf{c}_{\sigma}(x) = s) \\ (\!|Q|\!)^{\eta}_{\sigma}(s) & \text{otherwise} \end{cases}
\end{array}
$$

We can then define composition $\Vdash^{\sigma,\eta}_{\rho} \Xi; \Delta; \Gamma; \mathcal{G}$ as in Fig. 7 extending the conditions (a-f) considering d_{σ} and d_{η} as sets (and the appropriate checks for set membership and non-membership).

Lemma 7.3 (Partial Global Types). *A grammar of partial global types generated under Definitions 7.1 and 7.2 for each session is given as:*

$$
\begin{array}{rcl}
G & ::= & \mathsf{end} \mid \tilde{\mathsf{p}} \rightsquigarrow \mathsf{q} :!(T).\mathsf{end} \mid \mathsf{p}_1 \rightsquigarrow \mathsf{q} :?(T).\mathsf{p}_2 \rightsquigarrow \mathsf{q} :?(T).\dots\mathsf{p}_n \rightsquigarrow \mathsf{q} :?(T).\mathsf{end} \\
& \mid & \mathsf{p} \rightsquigarrow \mathsf{q}{:} \uparrow (T).G \mid \mathsf{p} \rightsquigarrow \mathsf{q}{:} \downarrow (T).G \mid \mathsf{p} \rightsquigarrow \mathsf{q}{:}\oplus\{l_j{:}G_j\}_{j\in J} \mid \mathsf{p} \rightsquigarrow \mathsf{q}{:}\ \&\ \{l_j{:}G_j\}_{j\in J}
\end{array}
$$

Proof. By Definition 7.1(1,3,4), if x in $!x(z).P$ is assigned to s, P does not contain any channels mapped to s. Hence if typable Q contains both $C_1[!x(z).P]$ (with type of x is !) and $C_2[\overline{x}\langle y\rangle.R]$ (with type of x is ?) as its subterms, (1) C_i does not contain replication mapped to s; (2) C_1 does not contain channels of ? type mapped to s; and (3) we can set C_2 so that it not contain channels of ? type mapped to s. Thus at MP session s, $\tilde{\mathsf{p}} \rightsquigarrow \mathsf{q} :!(T).\mathsf{end}$ is generated from $!x(z).P$, and $\mathsf{p}_1 \rightsquigarrow \mathsf{q} :?(T).\mathsf{p}_2 \rightsquigarrow \mathsf{q} :?(T).\dots\mathsf{p}_n \rightsquigarrow \mathsf{q} :?(T).\mathsf{end}$ with $\mathsf{p}_j \in \tilde{\mathsf{p}}$ is generated from $\overline{x}\langle y\rangle.R$ where the condition $\mathsf{p}_j \in \tilde{\mathsf{p}}$ is ensured by the conditions of $\Vdash^{\sigma,\eta}_{\rho}$. □

Fusing and complete global types. We can now define fuse over partial global types given in Lemma 7.3. The definition is extended as follows (with $T_1 \leq T_2$, $\mathsf{p} \in \tilde{\mathsf{p}}$ and omitting the congruence cases where the the fuse operation pushes under both the partial and complete replication prefixes):

$$
\begin{array}{rcll}
\mathsf{fuse}(\tilde{\mathsf{p}} \rightsquigarrow \mathsf{q} :!(T).\mathsf{end}, \mathsf{end}) & = & \tilde{\mathsf{p}} \to \mathsf{q} : *(T).\mathsf{end} & \\
\mathsf{fuse}(\tilde{\mathsf{p}} \to \mathsf{q} : *(T).\mathsf{end}, \mathsf{end}) & = & \tilde{\mathsf{p}} \to \mathsf{q} : *(T).\mathsf{end} & \\
\mathsf{fuse}(\tilde{\mathsf{p}} \rightsquigarrow \mathsf{q} :!(T_1).G_1, \mathsf{p} \rightsquigarrow \mathsf{q} :?(T_2).G_2) & = & \mathsf{fuse}(\tilde{\mathsf{p}} \rightsquigarrow \mathsf{q} :!(T_1).G_1, G_2) & \mathsf{p} \in \tilde{\mathsf{p}}
\end{array}
$$

Lemma 7.4 (Complete Global Types). *A grammar of complete global types generated under Definitions 7.1 and 7.2 with* fuse *for each session is given as:*

$$
G \quad ::= \quad \mathsf{end} \mid \tilde{\mathsf{p}} \to \mathsf{q} : *(T).\mathsf{end} \mid \mathsf{p} \to \mathsf{q}{:}(T).G \mid \mathsf{p} \to \mathsf{q}{:}\{l_j{:}G_j\}_{j\in J}
$$

Proof. We only need to check that the three rules for replication defined above produce a global type of the form $\tilde{\mathsf{p}} \to \mathsf{q} : *(T).\mathsf{end}$. By Lemma 7.3, the third rule is replaced by

$$
\mathsf{fuse}(\tilde{\mathsf{p}} \rightsquigarrow \mathsf{q} :!(T_1).\mathsf{end}, \mathsf{p} \rightsquigarrow \mathsf{q} :?(T_2).G_2) = \mathsf{fuse}(\tilde{\mathsf{p}} \rightsquigarrow \mathsf{q} :!(T_1).\mathsf{end}, G_2) \quad \mathsf{p} \in \tilde{\mathsf{p}}
$$

where G_2 is either end or $\mathsf{p}_1 \rightsquigarrow \mathsf{q} :?(T_2).\dots\mathsf{p}_n \rightsquigarrow \mathsf{q} :?(T_2).\mathsf{end}$ where $\mathsf{p}_i \in \tilde{\mathsf{p}}$. If $G_2 = \mathsf{end}$, then the next matched rule is the first rule, which produces $\tilde{\mathsf{p}} \to \mathsf{q} : *(T).\mathsf{end}$. We repeat the third rule until we reach $\mathsf{p}_n \rightsquigarrow \mathsf{q} :?(T_2).\mathsf{end}$ to then produce $\tilde{\mathsf{p}} \to \mathsf{q} : *(T).\mathsf{end}$. □

Note that the second rule of $\mathsf{fuse}(G)$ would be used when the generated global type is in the form of the second line of the grammar in Lemma 7.3.

Example 7.5 (Racing on a replicated session). We illustrate the concepts pertaining to replication by considering the following processes (as in earlier examples, we assume basic value passing):

$$\begin{array}{rcl} P & \triangleq & !x(y).y(z_1).y(z_2).y\langle z_1+z_2\rangle \\ Q & \triangleq & \overline{x}\langle y\rangle.y\langle 1\rangle.y\langle 2\rangle.y(z_3) \\ R & \triangleq & \overline{x}\langle y\rangle.y\langle 2\rangle.y\langle 1\rangle.y(z_4) \end{array}$$

where $P \vdash_{\mathsf{CL}} \cdot; x{:}!(\mathsf{Int} \parr \mathsf{Int} \parr \mathsf{Int} \otimes \mathbf{1})$, $Q \vdash_{\mathsf{CL}} \cdot; x{:}?(\mathsf{Int} \otimes \mathsf{Int} \otimes \bot)$ and $R \vdash_{\mathsf{CL}} \cdot; x{:}?(\mathsf{Int} \otimes \mathsf{Int} \otimes \bot)$. We define mappings σ, σ_1, σ_2 and η, η_1, η_2 such that:

$$\begin{array}{rcl} [\![P]\!]^{\eta}_{\sigma} & = & s[\mathsf{p}]{:}\{\mathsf{q},\mathsf{r}\}!(\mathsf{p}_0\!\uparrow\!(\mathsf{Int});\mathsf{p}_0\!\uparrow\!(\mathsf{Int});\mathsf{p}_0\!\downarrow\!(\mathsf{Int})) \\ [\![Q]\!]^{\eta_1}_{\sigma_1} & = & s[\mathsf{q}]{:}\mathsf{p}?(\mathsf{p}_0\!\uparrow\!(\mathsf{Int});\mathsf{p}_0\!\uparrow\!(\mathsf{Int});\mathsf{p}_0\!\downarrow\!(\mathsf{Int})) \\ [\![R]\!]^{\eta_2}_{\sigma_2} & = & s[\mathsf{r}]{:}\mathsf{p}?(\mathsf{p}_0\!\uparrow\!(\mathsf{Int});\mathsf{p}_0\!\uparrow\!(\mathsf{Int});\mathsf{p}_0\!\downarrow\!(\mathsf{Int})) \end{array}$$

Generating the following global types:

$$\begin{array}{rcl} (\!|P|\!)^{\eta}_{\sigma} & = & \{\mathsf{q},\mathsf{r}\} \rightsquigarrow \mathsf{q} : !(\mathsf{p}_0\!\uparrow\!(\mathsf{Int});\mathsf{p}_0\!\uparrow\!(\mathsf{Int});\mathsf{p}_0\!\downarrow\!(\mathsf{Int})).\mathsf{end} \\ (\!|Q|\!)^{\eta_1}_{\sigma_1} & = & \{\mathsf{q},\mathsf{r}\} \rightsquigarrow \mathsf{q} : ?(\mathsf{p}_0\!\uparrow\!(\mathsf{Int});\mathsf{p}_0\!\uparrow\!(\mathsf{Int});\mathsf{p}_0\!\downarrow\!(\mathsf{Int})).\mathsf{end} \\ (\!|R|\!)^{\eta_2}_{\sigma_2} & = & \{\mathsf{q},\mathsf{r}\} \rightsquigarrow \mathsf{q} : ?(\mathsf{p}_0\!\uparrow\!(\mathsf{Int});\mathsf{p}_0\!\uparrow\!(\mathsf{Int});\mathsf{p}_0\!\downarrow\!(\mathsf{Int})).\mathsf{end} \end{array}$$

It is then straightforward that

$$\mathsf{fuse}((\!|P|\!)^{\eta}_{\sigma}, \mathsf{fuse}((\!|Q|\!)^{\eta_1}_{\sigma_1}, (\!|R|\!)^{\eta_1}_{\sigma_1}) = \{\mathsf{q},\mathsf{r}\} \to \mathsf{p} : *(\mathsf{p}_0\!\uparrow\!(\mathsf{Int});\mathsf{p}_0\!\uparrow\!(\mathsf{Int});\mathsf{p}_0\!\downarrow\!(\mathsf{Int})).\mathsf{end}$$

Local types and liveness. To prove the main results, we now extend the syntax of local types with the constructs $\tilde{\mathsf{p}}!(T)$ and $\mathsf{p}?(T)$, denoting the type of a participant that expects to input from $\tilde{\mathsf{p}}$, afterwards spawning a replica of type T, and the type of a participant that will use a replicated session offered by p (by sending it a fresh session) of type T.

Similarly, the syntax of binary types T extends with $!(T)$ and $?(T)$ and the duality of binary types is extended to $\overline{!(T)} = ?(\overline{T})$ and $\overline{?(T)} = !(\overline{T})$. Then the partial projection is defined as an input and an output of Definition 2.4, respectively.

The projection of $G = \tilde{\mathsf{s}} \to \mathsf{r} : *(T).\mathsf{end}$ is defined as follows:

$$G\!\upharpoonright\!\mathsf{p} = \mathsf{r}?(T) \text{ if } \mathsf{p} \in \tilde{\mathsf{s}}; \quad G\!\upharpoonright\!\mathsf{p} = \tilde{\mathsf{s}}!(T) \text{ if } \mathsf{p} = \mathsf{r}; \quad G\!\upharpoonright\!\mathsf{p} = \mathsf{end} \text{ otherwise.}$$

The labels for LTSs are extended with $\mathsf{p}\tilde{\mathsf{q}}!(T)$ and $\mathsf{pq}?(T)$, and the LTS rules of the local types are defined as:

$$\mathsf{q}?(T) \xrightarrow{\mathsf{pq}?(T)} \mathsf{end} \qquad\qquad \tilde{\mathsf{q}}!(T) \xrightarrow{\mathsf{p}\tilde{\mathsf{q}}!(T)} \tilde{\mathsf{q}}!(T)$$

We extend the duality of labels as $\overline{\mathsf{p}\tilde{\mathsf{q}}!(T)} = \mathsf{q}_i\mathsf{p}?(T)$ and $\overline{\mathsf{q}_i\mathsf{p}?(T)} = \mathsf{p}\tilde{\mathsf{q}}!(T)$ with $\mathsf{q}_i \in \tilde{\mathsf{q}}$. Then the transitions of the configurations do not change. The semantics of G is defined as

$$(\mathsf{expo}) \quad \tilde{\mathsf{p}} \to \mathsf{q} : *(T).\mathsf{end} \xrightarrow{\mathsf{pq}?(T)\cdot\mathsf{q}\tilde{\mathsf{p}}!(T)} \tilde{\mathsf{p}} \to \mathsf{q} : *(T).\mathsf{end} \quad \text{with } \mathsf{p} \in \tilde{\mathsf{p}}$$

We extend Definition 4.7 by adding the following cases.

4. if $\ell = \mathsf{pq}?(T)$ there exists $C \xrightarrow{\vec{\ell'}} C' \xrightarrow{\ell}\xrightarrow{\overline{\ell}} C''$;
5. if $\ell = \mathsf{p}\tilde{\mathsf{q}}!(T)$ for all $C \xrightarrow{\vec{\ell'}} C''$ such that $C'' = (T'_{\mathsf{p}})_{\mathsf{p}\in\mathcal{P}}$, $T'_{\mathsf{p}} \xrightarrow{\ell} T''_{\mathsf{p}}$.

Theorem 4.9 is updated replacing (DF) by the following liveness property, which allows for leftover replicated types (akin to Definition 2.2). Note that a difference from Theorem 4.9 is that ?-output is treated as an output and a selection since the replication is always available.

Definition 7.6 (Live). $C = (T_{0\mathrm{p}})_{\mathrm{p}\in\mathcal{P}}$ is *live* if for all $C \xrightarrow{\vec{\ell}} C_1 = (T_{\mathrm{p}})_{\mathrm{p}\in\mathcal{P}}$, if $T_{\mathrm{p}} \xrightarrow{\ell} T'_{\mathrm{p}}$ and ℓ is not !, there exists $C' = (T''_{\mathrm{p}})_{\mathrm{p}\in\mathcal{P}}$ such that $C_1 \xrightarrow{\vec{\ell}'} C'$ and (1) $C' \xrightarrow{\ell\cdot\overline{\ell}} C''$ if ℓ is an output or a selection or ?-output; (2) $C' \xrightarrow{\overline{\ell}\cdot\ell} C''$ if ℓ is an input; or (3) $C' \xrightarrow{\overline{\ell}'\cdot\ell'} C''$ if $\ell = \mathrm{pq} \triangleright l$ with some $\ell' = \mathrm{pq} \triangleright l'$. ⋄

For the main theorem, we replace Definition 4.8 by Definition 7.6 to account for replication by mirroring the notion of live process.

Theorem 7.7. *Let $P \Vdash^{\emptyset,\eta}_{\rho} \cdot;\Delta;\Gamma;\mathcal{G}$ and $\Delta = \emptyset$ or Δ contains only* $\mathbf{1}$ *or* $\perp$. *We have:* (1) *If* $live(P)$ *then* $P \to P'$*;* (2) *at each session* s, $\mathrm{fuse}_s(\mathcal{G})$ *is well-formed and live; and* (3) *the ING is acyclic.*

Proof. (1) follows from [9, 60]; (2) we first note that replication is mapped to distinct multiparty sessions. Lemma 7.4 proves well-formedness. By Lemma 7.3 and Lemma 7.4, a complete replicated global type at each s is the form of $\tilde{\mathrm{p}} \to \mathrm{q} : *(T).\mathrm{end}$. Then by the rules of LTSs defined above, if $\ell = \mathrm{pq}?(T)$ (?-output, which is the case of (1) in Definition 7.6), then $C' \xrightarrow{\ell\cdot\overline{\ell}} C''$ since C contains $\mathrm{p}\tilde{\mathrm{q}}!(T)$ by the definition of projection. Hence for each s, $\mathrm{fuse}_s(\mathcal{G})$ is live; and (3) follows the same argument as Theorem 5.7, noting that each replicated session corresponds to a different session. □

The uniqueness theorem for replication, cf. Theorems 3.9 and 6.10, can be obtained under the condition where each replicated channel is assigned to a new session.

Example 7.8. The following processes (from an example of [14]) are untypable in the global progress type system of [18].

$$P \triangleq\, !x(y).y(n) \vdash_{\mathsf{CL}} x{:}!(\mathsf{int} \parr \mathbf{1}) \quad Q \triangleq\, !z(w).w\langle 1\rangle \vdash_{\mathsf{CL}} z{:}!(\mathsf{int} \otimes \mathbf{1})$$
$$R \triangleq \overline{x}\langle y\rangle.\overline{z}\langle w\rangle.w(n).y\langle n\rangle.\mathbf{0} \vdash_{\mathsf{CL}} \cdot; x{:}?(\mathsf{int} \otimes \perp), z{:}?(\mathsf{int} \parr \perp)$$

We define mappings $\sigma, \sigma_1, \sigma_2$ and η, η_1, η_2 such that:

$$(\!|P|\!)^{\eta}_{\sigma} = s[\mathrm{p}]{:}\mathrm{r}!(\mathrm{p}_0{\uparrow}(\mathsf{int})) \quad (\!|Q|\!)^{\eta_1}_{\sigma_1} = s'[\mathrm{q}]{:}\mathrm{b}!(\mathrm{r}{\downarrow}(\mathsf{int})) \quad (\!|R|\!)^{\eta_2}_{\sigma_2} = s[\mathrm{r}]{:}\mathrm{p}?(\mathrm{p}_0{\uparrow}(\mathsf{int})), s'[\mathrm{b}]{:}\mathrm{q}?(\mathrm{r}{\downarrow}(\mathsf{int}))$$

Then $(\boldsymbol{\nu}x,z)(P \mid Q \mid R) \Vdash \cdot;\cdot; (\!|P|\!)^{\eta}_{\sigma}, (\!|Q|\!)^{\eta_1}_{\sigma_1}, (\!|R|\!)^{\eta_2}_{\sigma_2}$.

8 MULTICUT IN CLL

The inability to compose processes that interact by sharing more than one channel – often dubbed *multicut* – significantly limits the interconnection networks in CLL. Logically, such a form of unrestricted multicut is *unsound*, and operationally results in deadlocks. Consider the following CLL processes:

$$P \triangleq y(x).z\langle 7\rangle.\mathbf{0} \qquad R \triangleq w(x).y\langle ''a''\rangle.\mathbf{0} \qquad Q_1 \triangleq z(x).w\langle \mathsf{tt}\rangle.\mathbf{0} \qquad Q_2 \triangleq w\langle \mathsf{tt}\rangle.z(x).\mathbf{0}$$

We have that P is typed in a context $\Delta = y{:}\mathsf{str} \parr \perp, z{:}\mathsf{int} \otimes \mathbf{1}$, R is typed in a context $\Delta' = y{:}\mathsf{str} \otimes \mathbf{1}, w{:}\mathsf{bool} \parr \perp$ and both Q_1 and Q_2 are typed in a context $\Delta'' = z{:}\mathsf{int} \parr \perp, w{:}\mathsf{bool} \otimes \mathbf{1}$. We can observe that the composition $(\boldsymbol{\nu}y,z,w)\ (P|Q_1|R)$ is clearly deadlocked (viz. § 2.2). However, the composition $(\boldsymbol{\nu}y,z,w)(P|Q_2|R)$ is safe, with the processes reducing in three steps to $\mathbf{0}$. Intuitively, despite Q_1 and Q_2 both implementing the same sessions, their (identical) typings do not distinguish the sequential orderings of actions.

As discussed in § 3, the framework of MP can distinguish such orderings. Thus we may appeal to the higher discriminating power of PMC in order to eliminate these unsafe (i.e. deadlocking) multicuts. For instance, the following MP processes can be mapped from the CLL processes above:

$$\begin{array}{rcl@{\qquad}rcl} P' & \triangleq & s[\mathrm{p}][\mathrm{r}](x); s[\mathrm{p}][\mathrm{q}]\langle 7\rangle & R' & \triangleq & s[\mathrm{r}][\mathrm{q}](x); s[\mathrm{r}][\mathrm{p}]\langle ''a''\rangle \\ Q'_1 & \triangleq & s[\mathrm{q}][\mathrm{p}](x); s[\mathrm{q}][\mathrm{r}]\langle \mathsf{tt}\rangle & Q'_2 & \triangleq & s[\mathrm{q}][\mathrm{r}]\langle \mathsf{tt}\rangle; s[\mathrm{q}][\mathrm{p}](x) \end{array}$$

$$\text{(MCut)}\ \frac{P \Vdash^{\sigma}_{\rho} \Delta, x_1{:}A_1, \ldots, x_n{:}A_n; \mathcal{G}_1 \quad Q \Vdash^{\sigma'}_{\rho'} \Delta', x_1{:}A_1^{\perp}, \ldots, x_n{:}A_n^{\perp}; \mathcal{G}_2 \quad \mathcal{G} = \mathsf{fuse}(\mathcal{G}_1, \mathcal{G}_2) \text{ defined} \quad \rho' \cap \rho = \emptyset \quad (\ddagger)}{(\boldsymbol{\nu} x_1, \ldots, x_n)(P \mid Q) \Vdash^{(\sigma' \cup \sigma)\setminus\{x_1,..,x_n\}}_{\rho''} \Delta, \Delta'; \mathcal{G}}$$

($\ddagger$) **(a)** bound channels: $\rho'' = \rho \cup \rho' \cup_i (x_i, s[\mathsf{p}_\sigma(x_i)][\mathsf{d}_\sigma(x_i)])$
(b) role/destination match: $\mathsf{p}_\sigma(x_i) = \mathsf{d}_{\sigma'}(x_i) \wedge \mathsf{d}_\sigma(x_i) = \mathsf{p}_{\sigma'}(x_i)$
(c) unique destination: $\forall y \in \Delta. z \in \Delta'.(\mathsf{d}_\sigma(y), \mathsf{d}_{\sigma'}(z) \notin \rho, \rho' \wedge$
(d) role/destination disjointness: $\mathsf{p}_\sigma(y) \neq \mathsf{p}_{\sigma'}(z) \wedge \mathsf{p}_\sigma(y) \neq \mathsf{d}_{\sigma'}(z) \wedge \mathsf{p}_{\sigma'}(z) \neq \mathsf{d}_\sigma(y))$

Fig. 8. Multicut Rule

Then the (partial) global types generated from P', R' and Q'_1 are *not* PMC, whereas those from P', R' and Q'_2 are:

$$\begin{array}{lllll} (\!|P'|\!) & \triangleq & \mathsf{r} \rightsquigarrow \mathsf{p}{:} \downarrow (\mathsf{str}).\mathsf{p} \rightsquigarrow \mathsf{q}{:} \uparrow (\mathsf{int}).\mathsf{end} & (\!|Q'_1|\!) \triangleq & \mathsf{p} \rightsquigarrow \mathsf{q}{:} \downarrow (\mathsf{int}).\mathsf{q} \rightsquigarrow \mathsf{r}{:} \uparrow (\mathsf{bool}).\mathsf{end} \\ (\!|R'|\!) & \triangleq & \mathsf{q} \rightsquigarrow \mathsf{r}{:} \downarrow (\mathsf{bool}).\mathsf{r} \rightsquigarrow \mathsf{p}{:} \uparrow (\mathsf{str}).\mathsf{end} & (\!|Q'_2|\!) \triangleq & \mathsf{q} \rightsquigarrow \mathsf{r}{:} \uparrow (\mathsf{bool}).\mathsf{p} \rightsquigarrow \mathsf{q}{:} \downarrow (\mathsf{int}).\mathsf{end} \end{array}$$

We thus make use of PMC to develop two deadlock-free multicut rules for CLL. We allow processes to share multiple channels holding dual types of each other (as the multicut rule in [2]) but restrict composition by requiring the induced partial global types to be *fuseable* (or PMC), recovering deadlock-freedom.

Definition 8.1 (Multicut – No Name Passing or Replication). Using the mapping of § 3, we redefine the judgement $P \Vdash^{\sigma}_{\rho} \Delta; \mathcal{G}$ to produce a multicut rule, defined in Fig. 8 where "$\mathsf{fuse}(\mathcal{G}_1, \mathcal{G}_2)$ defined" means that for all $G_i \in \mathcal{G}_1 \cup \mathcal{G}_2$, $\mathsf{fuse}(\cdots (\mathsf{fuse}(G_1, G_2), G_3)) \cdots, G_n)$ is defined (Definition 4.10).

Rule (MCut) is symmetric, matching a generalised cut rule. We highlight the differences of (MCut) from (comp) and (comp$_d$) with red letters. Clauses **(a)** and **(b)** are fundamentally unchanged from (comp), requiring the mappings for composed channels to match. We note $\mathsf{c}_\sigma(x) = \mathsf{c}_{\sigma'}(y) = s$ for all free names x and y since we consider a single session. Clauses **(c)** and **(d)** are as in (comp$_d$), but we remove the condition $\mathsf{d}_\sigma(z) \neq \mathsf{d}_{\sigma'}(y)$ to enable multicut.

Theorem 8.2. *Let $P \Vdash^{\sigma}_{\rho} \Delta; \Gamma; \mathcal{G}$ and $\Delta = \emptyset$ or Δ contains only* $\mathbf{1}$ *or* $\perp$. *Then:* (1) $P \rightarrow^* \mathbf{0}$*; and* (2) $\mathsf{fuse}(\mathcal{G})$ *is a complete global type.*

Proof. (1) By the same proposition as Proposition 3.6, we obtain if $P \Vdash^{\sigma}_{\rho} \Delta; \Gamma$, then $\rho(\sigma(P)) \vdash_{\mathsf{MP}} \Gamma$. By the definition, Γ only contains a single multiparty session where each prefix is simple [31, Definition 5.25 in JACM]. Since $\Delta = \emptyset$ or Δ contains only $\mathbf{1}$ or $\perp$, Γ is coherent. By the result of [31, § 5], $\rho(\sigma(P)) \rightarrow^* \mathbf{0}$. Then by the operational correspondence between MP and CLL, $P \rightarrow^* \mathbf{0}$. (2) By construction of (MCut). □

Termination (Theorem 8.2(1)) follows by the fact that the calculus preserves a single (recursion-free) multiparty session [31, § 5]; The fact that multicut of a complete session generates a complete global type (Theorem 8.2(2)) is by construction. While forms of multicut are in general logically unsound since they invalidate cut elimination [60, § 6], our (MCut) rule ensures that closed proof terms that may be composed do indeed have progress (and terminate – implying soundness).

We note that in the presence of general delegation (and thus interleaved multiparty sessions) multicut would construct a well-typed (and fuseable) composition of processes that deadlock. However, using the partial type generation in § 6, we can formulate a simple version of multicut

$$\text{(MCut)}\ \frac{P \vDash^{\sigma,\eta}_{\rho} \Delta, x_1{:}A_1, \ldots, x_n{:}A_n; \mathcal{G}_1 \quad Q \vDash^{\sigma',\eta'}_{\rho'} \Delta', x_1{:}A_1^{\perp}, \ldots, x_n{:}A_n^{\perp}; \mathcal{G}_2 \quad \mathsf{fuse}(\mathcal{G}_1,\mathcal{G}_2)\ \text{defined} \quad \rho' \cap \rho = \emptyset \quad (\ddagger)}{(\boldsymbol{\nu} x_1, \ldots, x_n)(P \mid Q) \vDash^{(\sigma'\cup\sigma)\setminus\{x_1,..,x_n\},\eta\cup\eta'}_{\rho''} \Delta, \Delta'; \mathsf{fuse}(\mathcal{G}_1,\mathcal{G}_2)}$$

($\ddagger$) **(a)** bound channels: $\rho'' = \rho \cup \rho' \cup_i (x_i, s[\mathsf{p}_\sigma(x_i)][\mathsf{d}_\sigma(x_i)])$
(b) role/destination match: $\mathsf{c}_\sigma(x_i) = \mathsf{c}_{\sigma'}(x_i) \quad \mathsf{p}_\sigma(x_i) = \mathsf{d}_{\sigma'}(x_i) \quad \mathsf{d}_\sigma(x_i) = \mathsf{p}_{\sigma'}(x_i)$
(c) unique destination: $\forall y \in \Delta. z \in \Delta'.(\mathsf{d}_\sigma(y), \mathsf{d}_{\sigma'}(z) \notin \rho, \rho' \wedge$
(d) role/destination disjointness: $\mathsf{p}_\sigma(y) \neq \mathsf{p}_{\sigma'}(z) \wedge\ \mathsf{p}_\sigma(y) \neq \mathsf{d}_{\sigma'}(z) \wedge\ \mathsf{p}_{\sigma'}(z) \neq \mathsf{d}_\sigma(y))$
(e) bound role/destination match: $\forall x \in \eta. \forall y \in \eta'. x = y \Rightarrow \mathsf{d}_\eta(x) = \mathsf{p}_{\eta'}(y) \vee \mathsf{p}_\eta(x) = \mathsf{d}_{\eta'}(y)$
(f) bound channel disjointness: (1) $\forall x \in \eta. \forall y \in \varphi = \{\sigma', \eta', \rho, \rho'\}. \mathsf{c}_\eta(x) \neq \mathsf{c}_\varphi(y)$
(2) $\forall x \in \eta'. \forall y \in \varphi = \{\sigma', \eta, \rho, \rho'\}. \mathsf{c}_{\eta'}(x) \neq \mathsf{c}_\varphi(y)$

Fig. 9. Multicut with Channel Passing

with channel passing, by restricting partial global type generation (Definition 6.7) to only be defined when sent channels are consistently assigned within the same MP session, as defined below.

Definition 8.3 (Multicut with Name Passing). Using the mapping of § 6 but where all free names are mapped to the same session, by restricting partial global type generation (Definition 6.7) to only be defined when in the clause for channel output, $\forall s \in \sigma, \eta. (\!|Q_1|\!)^{\eta}_{\sigma}(s) = (\!|Q_1|\!)^{\eta}_{\sigma}(\eta(y))$, we define a multicut rule for name passing in Fig. 9.

The above (MCut) extends the rule of Fig. 8 with clauses **(e)-(f)** from Fig. 7 (red letters highlight differences), which ensure that channels that are to be exchanged are mapped to fresh channels with consistent role assignments in MP.

The same deadlock-freedom as in Theorem 8.2 can be obtained given that the (MCut) rule of Fig. 7 stays within a single session.

We refrain from introducing a multicut for replication since ($\mathsf{cut}^!$) already enables us to share replicated sessions among many clients. Moreover, a cut-free process can only implement a single replicated session (i.e. $P \nvDash_{\mathrm{CL}} \Xi; x{:}!A, y{:}!B$ cut-free).

Example 8.4 (Two Buyer Protocol [31]). The following defines the coordination of two Buyers seeking to buy from a Seller.

$$\begin{aligned}
\mathsf{Seller} &\triangleq x(t).x\langle 32\rangle.y\langle 32\rangle.y.\mathsf{case}\{ok{:}y(s).\mathbf{0}, nok{:}\mathbf{0}\}\\
\mathsf{Buyer}_1 &\triangleq x\langle\text{"xpto"}\rangle.x(n).z\langle n/2\rangle.\mathbf{0}\\
\mathsf{Buyer}_2 &\triangleq y(n).z(m).y.ok; y\langle\text{"foo"}\rangle.\mathbf{0}
\end{aligned}$$

The Seller uses channels x and y to interact with Buyer_1 and Buyer_2. Buyer_1 uses channel z to interact with Buyer_2. The Seller waits for Buyer_1 to send it the title of the item that is to be purchased, replying to both buyers with a price. Buyer_1 then emits to Buyer_2 how much it will contribute in the purchase. Buyer_2 then chooses to agree and sends to the Seller an *ok* message and a string (e.g. the shipping address). Consider $\sigma, \sigma_1, \sigma_2$:

$$\begin{aligned}
\sigma(\mathsf{Seller}) &= s[\mathrm{s}][\mathrm{b1}](t); s[\mathrm{s}][\mathrm{b1}]\langle 32\rangle; s[\mathrm{s}][\mathrm{b2}]\langle 32\rangle; s[\mathrm{s}][\mathrm{b2}] \,\&\, \{ok{:}s[\mathrm{s}][\mathrm{b2}](s), nok{:}\mathbf{0}\}\\
\sigma_1(\mathsf{Buyer}_1) &= s[\mathrm{b1}][\mathrm{s}]\langle\text{"xpto"}\rangle; s[\mathrm{b1}][\mathrm{s}](n); s[\mathrm{b1}][\mathrm{b2}]\langle n/2\rangle; \mathbf{0}\\
\sigma_2(\mathsf{Buyer}_2) &= s[\mathrm{b2}][\mathrm{s}](n); s[\mathrm{b2}][\mathrm{b1}](m); s[\mathrm{b2}][\mathrm{s}] \oplus ok; s[\mathrm{b2}][\mathrm{s}]\langle\text{"foo"}\rangle; \mathbf{0}
\end{aligned}$$

In the development of § 5 the processes above would *not* be composable. We may now compose them via multicut:

$$
\begin{array}{rcl}
(\!|\mathsf{Seller}|\!)_\sigma & = & \mathsf{b1} \rightsquigarrow \mathsf{s} :\downarrow (\mathsf{str}).\mathsf{s} \rightsquigarrow \mathsf{b1} :\uparrow (\mathsf{int}).\mathsf{s} \rightsquigarrow \mathsf{b2} :\uparrow (\mathsf{int}). \\
& & \mathsf{b2} \rightsquigarrow \mathsf{s} : \&\{ok{:}\mathsf{b2} \rightsquigarrow \mathsf{s} :\downarrow (\mathsf{str}).\mathsf{end}, nok{:}\mathsf{end}\} \\
(\!|\mathsf{Buyer}_1|\!)_{\sigma_1} & = & \mathsf{b1} \rightsquigarrow \mathsf{s} :\uparrow (\mathsf{str}).\mathsf{s} \rightsquigarrow \mathsf{b1} :\downarrow (\mathsf{int}).\mathsf{b1} \rightsquigarrow \mathsf{b2} :\uparrow (\mathsf{int}).\mathsf{end} \\
(\!|\mathsf{Buyer}_2|\!)_{\sigma_2} & = & \mathsf{s} \rightsquigarrow \mathsf{b2} :\downarrow (\mathsf{int}).\mathsf{b1} \rightsquigarrow \mathsf{b2} :\downarrow (\mathsf{int}).\mathsf{b2} \rightsquigarrow \mathsf{s} : \oplus\{ok : \mathsf{b2} \rightsquigarrow \mathsf{s} :\uparrow (\mathsf{int}).\mathsf{end}\}
\end{array}
$$

Let $G_1 = (\!|\mathsf{Seller}|\!)_\sigma$, $G_2 = (\!|\mathsf{Buyer}_1|\!)_{\sigma_1}$ and $G_3 = (\!|\mathsf{Buyer}_2|\!)_{\sigma_2}$. Then we have that the fuse of the three global types is:

$$
\mathsf{b1} \to \mathsf{s} : (\mathsf{str}).\mathsf{s} \to \mathsf{b1} : (\mathsf{int}).\mathsf{s} \to \mathsf{b2} : (\mathsf{int}).\mathsf{b1} \to \mathsf{b2} : (\mathsf{int}).\mathsf{b2} \to \mathsf{s} : \{ok : \mathsf{b2} \to \mathsf{s} : (\mathsf{str}).\mathsf{end}\}
$$

Note that we do not require any modifications to the syntax of CLL processes to represent and verify this multiparty example.

Example 8.5 (Multicut and channel passing). We illustrate a multicut with channel passing via the following processes:

$$
\begin{array}{rcl}
P & \triangleq & \overline{x}\langle y\rangle.(y\langle 1\rangle.y(z_1) \mid z\langle 2\rangle.x(z_2)) \\
Q & \triangleq & x(y).y(z_4).x\langle 3\rangle.y\langle 4\rangle.w\langle 5\rangle \\
R & \triangleq & z(z_3).w(z_4)
\end{array}
$$

where $P \vdash_{\mathsf{CL}} x{:}(\mathsf{Int} \otimes \mathsf{Int} \parr \mathbf{1}) \otimes (\mathsf{Int} \parr \bot), z{:}\mathsf{Int} \otimes \mathbf{1}$, $Q \vdash_{\mathsf{CL}} x{:}(\mathsf{Int} \parr \mathsf{Int} \otimes \bot) \parr (\mathsf{Int} \otimes \mathbf{1}), w{:}\mathsf{Int} \otimes \bot$ and $R \vdash_{\mathsf{CL}} z{:}\mathsf{Int} \parr \bot, w{:}\mathsf{int} \parr \mathbf{1}$ We define mappings $\sigma, \sigma_1, \sigma_2, \eta, \eta_1, \eta_2$ such that:

$$
\begin{array}{rcl}
\sigma(\eta(P)) & = & (\nu s')s[\mathsf{p}][\mathsf{q}]\langle s'[\mathsf{q}_1]\rangle; (s'[\mathsf{p}_1][\mathsf{q}_1]\langle 1\rangle; s'[\mathsf{p}_1][\mathsf{q}_1](z_1) \mid s[\mathsf{p}][\mathsf{r}]\langle 2\rangle; s[\mathsf{p}][\mathsf{q}](z_2)) \\
\sigma_1(\eta_1(Q)) & = & s[\mathsf{q}][\mathsf{p}](y); y[\mathsf{p}_1](z_4); s[\mathsf{q}][\mathsf{p}]\langle 3\rangle; y[\mathsf{p}_1]\langle 4\rangle; s[\mathsf{q}][\mathsf{r}]\langle 5\rangle \\
\sigma_2(\eta_2(R)) & = & s[\mathsf{r}][\mathsf{p}](z_3); s[\mathsf{r}][\mathsf{q}](z_4)
\end{array}
$$

We then have the following global types:

$$
\begin{array}{rcl}
(\!|P|\!)^\eta_\sigma & = & \mathsf{p} \rightsquigarrow \mathsf{q} :\uparrow (\mathsf{q}_1{\downarrow}(\mathsf{Int}); \mathsf{q}_1{\uparrow}(\mathsf{Int})).\mathsf{p} \rightsquigarrow \mathsf{r} :\uparrow (\mathsf{Int}).\mathsf{q} \rightsquigarrow \mathsf{p} :\downarrow (\mathsf{Int}).\mathsf{end} \\
(\!|Q|\!)^{\eta_1}_{\sigma_1} & = & \mathsf{p} \rightsquigarrow \mathsf{q} \downarrow (\mathsf{q}_1{\downarrow}(\mathsf{Int}); \mathsf{q}_1{\uparrow}(\mathsf{Int})).\mathsf{q} \rightsquigarrow \mathsf{p} :\uparrow (\mathsf{Int}).\mathsf{q} \rightsquigarrow \mathsf{r} :\uparrow (\mathsf{Int}).\mathsf{end} \\
(\!|R|\!)^{\eta_2}_{\sigma_2} & = & \mathsf{p} \rightsquigarrow \mathsf{r} :\downarrow (\mathsf{Int}).\mathsf{q} \rightsquigarrow \mathsf{r} :\downarrow (\mathsf{Int}).\mathsf{end}
\end{array}
$$

We have that: $\mathsf{fuse}((\!|P|\!)^\eta_\sigma, \mathsf{fuse}((\!|Q|\!)^{\eta_1}_{\sigma_1}, (\!|R|\!)^{\eta_2}_{\sigma_2})) = \mathsf{p} \to \mathsf{q} : (\mathsf{q}_1{\downarrow}(\mathsf{Int}); \mathsf{q}_1{\uparrow}(\mathsf{Int})).\mathsf{p} \to \mathsf{r} : (\mathsf{Int}).\mathsf{q} \to \mathsf{p} : (\mathsf{Int}).\mathsf{q} \to \mathsf{r} : (\mathsf{Int}).\mathsf{end}$, and so the multicut rule can be applied, validating the deadlock-free composition of P, Q and R.

9 DISCUSSION AND RELATED WORK

Interconnectability. The work [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 interconnectability (and deadlock-freedom in general) more challenging. The work [2] shows that compact-closed categories can interpret cyclic networks typed with multicut rules, but are unable to ensure deadlock-freedom. A categorical model of deadlock-free multicut (i.e. finding additional structure to interpret PMC) is interesting future work. The work [3] defines a multicut rule for a variant of CP [60] where propositions are self-dual. This work focuses on capturing the full power of the π-calculus and thus is not concerned with either deadlock-freedom (which their multicut rule does not ensure) or interconnectability.

Multiparty Sessions and Linear Logic. The works of [11, 14] and [7] are the most related to our own. The work [14] proposed a typing system for CLL extended to multiparty session primitives. The methodology follows a coherence-based multiparty session type framework, starting from a special form of global types which are annotated by linear logic modalities. The global types are projected to propositions annotated by roles, which type each CLL process. The multicut rule is applied to a complete set of processes in a multiparty session, directly using information of that global type. The work [11] develops a variation on the coherence-based approach, based on the work of [60] and [7], giving an interpretation of $\otimes$ and $\parr$ that is dual to that of [14]. We highlight that the work of [13] also applies a coherence-based linear logic interpretation to a choreographic (global) *language* which differs from multiparty session *types*.

Our work differs from the above approaches in several respects: (1) the work in [11, 13, 14] does not (aim to) study interconnectability; (2) the approach of [11, 14] relies on a special form of global types annotated by modalities, or propositions annotated by participants. Our compositional PMC-based approach requires no change to the semantics or syntax of processes, types, nor typing rules (except multicut) of CLL; (3) the multicut rule of [11, 14] 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 coherence-based cut rule [11, 14] is more limited than our synthesis-based approach. This is because a complete global type built by PMC characterises all deadlock-free traces observable from local type configurations (Theorem 4.9). For example, our system types Example 6.9, which is untypable in [11, 14] with the corresponding global type. Thus, our framework allows for more typable representatives of individual multiparty sessions.

The work [7] shows that the session interpretation of intuitionistic linear logic can encode the behaviour of multiparty sessions (up to typed barbed congruence). While the goals of our work are quite different, focusing on interconnection networks of CLL processes, we note that their work does not contradict with our results. Their encoding consists of adding *another* participant (i.e. an orchestrator) that mediates all interactions between roles (this encoding also appears in [11]). Thus, a network such as that of Fig. 1a is realised by disconnecting all participants, adding a new (fourth) participant p', and connecting each participant only with p' (i.e. a tree topology). Our encoding preserves the interconnectability of global types, whereas the encoding in [7, 11] does not.

Degree of Sharing and Distributability. The work [19] identifies classes of deadlock free processes defined by the number of shared binary sessions between the parallel processes. Our connectability based on multiparty differs from their characterisation and is unrelated to the number of sessions in the process syntax: in our framework, two parallel MP binary processes and CLL have the same connectability since both calculi allow bidirectional interactions ($\mathsf{p} \leftrightarrow \mathsf{q}$), while in [19], CLL is strictly less expressive than two binary session processes with two shared channels. Notice that the work of [19] does not study replication or extensions of CLL with multicut as we have done in § 7 and § 8. An encoding criterion for synchronisation among parallel processes called *distributability* is studied in [52]. Their untyped criterion is not applicable to our setting since processes with the same interconnection network might not have the same distributability (and vice-versa). For instance, consider:

$$\begin{array}{rcl} R & \triangleq & s[\mathsf{r}][\mathsf{p}](y); s'[\mathsf{r}_0][\mathsf{q}](w); w[\mathsf{q}_0](n); y[\mathsf{p}_0]\langle n\rangle \\ R' & \triangleq & s[\mathsf{r}][\mathsf{p}](y); y[\mathsf{p}_1]\langle n\rangle \mid s'[\mathsf{r}_0][\mathsf{q}](w); w[\mathsf{q}_1](n) \end{array}$$

In R, $\mathsf{r} \leftrightarrow \mathsf{p}$ at s and $\mathsf{r}_0 \leftrightarrow \mathsf{q}$ at s' are independent, so that R' has the same interconnection structure as R. However, R has 1-distributability while R' has 2-distributability. So, results based on distributability do not in general imply ours.

Progress and Multiparty Compatibility. Type systems for progress in concurrent processes are a vast area of research. See, e.g. [18, 34, 50, 59]. While the main emphasis of this work is not a typing system for progress, our encodings ensure progress in restricted interleaved multiparty sessions.

Multiparty Compatibility (MC) properties are studied in [22, 37] where a global type is synthesised from communicating automata, assuming *all* participants are initially present. These global synthesis methods are not directly applicable to CLL where composition is binary. To overcome this issue, we proposed PMC together with fusing (partial) global types generated from CLL processes. Investigations of partial compatibility for choreographies [37] and timers [6] would allow us to capture larger classes of connectability (with timing information) in the CLL framework.

REFERENCES

[1] Samson Abramsky. 1993. Computational Interpretations of Linear Logic. *Theoret. Comput. Sci.* 111, 1–2 (1993), 3–57.

[2] Samson Abramsky, Simon J. Gay, and Rajagopal Nagarajan. 1995. Specification Structures and Propositions-as-Types for Concurrency. In *Logics for Concurrency*. 5–40.

[3] Robert Atkey, Sam Lindley, and J. Garrett Morris. 2016. Conflation Confers Concurrency. In *A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday (LNCS)*, Vol. 9600. Springer, 32–55.

[4] Gianluigi Bellin and Phil Scott. 1994. On the π-Calculus and Linear Logic. *Theoret. Comput. Sci.* 135, 1 (1994), 11–65.

[5] Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, and Nobuko Yoshida. 2017. Monitoring Networks through Multiparty Session Types. *TCS* 669 (2017), 33–58.

[6] Laura Bocchi, Julien Lange, and Nobuko Yoshida. 2015. Meeting Deadlines Together. In *CONCUR 2015 (LIPIcs)*, Vol. 42. Schloss Dagstuhl, 283–296.

[7] Luís Caires and Jorge A. Pérez. 2016. Multiparty Session Types within a Canonical Binary Theory, and Beyond. In *FORTE'16 (LNCS)*, Vol. 9688. Springer, 74–95.

[8] Luís Caires and Frank Pfenning. 2010. Session Types as Intuitionistic Linear Propositions. In *CONCUR (LNCS)*, Vol. 6269. Springer, 222–236.

[9] Luís Caires, Frank Pfenning, and Bernardo Toninho. 2016. Linear logic propositions as session types. *Math. Struct. in Comp. Sci.* 26, 3 (2016), 367–423.

[10] Sara Capecchi, Ilaria Castellani, and Mariangiola Dezani-Ciancaglini. 2016. Information flow safety in multiparty sessions. *Mathematical Structures in Computer Science* 26, 8 (2016), 1352–1394.

[11] Marco Carbone, Sam Lindley, Fabrizio Montesi, Carsten Schürmann, and Philip Wadler. 2016. Coherence Generalises Duality: a logical explanation of multiparty session types. In *CONCUR'16 (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 59. Schloss Dagstuhl, 33:1–33:15.

[12] Marco Carbone and Fabrizio Montesi. 2013. Deadlock-freedom-by-design: multiparty asynchronous global programming. In *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*. 263–274.

[13] Marco Carbone, Fabrizio Montesi, and Carsten Schürmann. 2014. Choreographies, Logically. In *CONCUR (LNCS)*, Vol. 8704. Springer, 47–62.

[14] Marco Carbone, Fabrizio Montesi, Carsten Schürmann, and Nobuko Yoshida. 2015. Multiparty Session Types as Coherence Proofs. In *CONCUR 2015 (LIPIcs)*, Vol. 42. Schloss Dagstuhl, 412–426.

[15] Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. 2017. Concurrent Reversible Sessions. In *CONCUR (LIPIcs)*, Vol. 85. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 30:1–30:17.

[16] Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Jorge A. Pérez. 2016. Self-adaptation and secure information flow in multiparty communications. *Formal Asp. Comput.* 28, 4 (2016), 669–696.

[17] Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, Alceste Scalas, and Nobuko Yoshida. 2017. On the Preciseness of Subtyping in Session Types. *Logical Methods in Computer Science* 13, 2 (2017).

[18] Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, and Luca Padovani. 2016. Global Progress for Dynamically Interleaved Multiparty Sessions. *MSCS* 26, 2 (2016), 238–302.

[19] Ornela Dardha and Jorge A. Pérez. 2015. Comparing Deadlock-Free Session Typed Processes. In *Express (EPTCS)*, Vol. 190. 1–15.

[20] Romain Demangeon, Kohei Honda, Raymond Hu, Rumyana Neykova, and Nobuko Yoshida. 2015. Practical Interruptible Conversations: Distributed Dynamic Verification with Multiparty Session Types and Python. *Formal Methods in System Design* 46, 3 (2015), 197–225.

[21] Romain Demangeon and Nobuko Yoshida. 2015. On the Expressiveness of Multiparty Sessions. In *FSTTCS'15 (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 45. Schloss Dagstuhl, 560–574.

[22] Pierre-Malo Deniélou and Nobuko Yoshida. 2013. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. In *ICALP'13 (LNCS)*, Vol. 7966. Springer, 174–186.

[23] Henry DeYoung, Luis Caires, Frank Pfenning, and Bernardo Toninho. 2012. Cut Reduction in Linear Logic as Asynchronous Session-Typed Communication. In *Computer Science Logic*.

[24] Luca Fossati, Raymond Hu, and Nobuko Yoshida. 2014. Multiparty Session Nets. In *TGC'14 (LNCS)*, Matteo Maffei and Emilio Tuosto (Eds.), Vol. 8902. Springer, 112–127.

[25] Simon Fowler. 2016. An Erlang Implementation of Multiparty Session Actors. In *ICE '16 (EPTCS)*, Vol. 223. 36–50.

[26] S. Gay and M. Hole. 2005. Subtyping for Session Types in the Pi Calculus. *Acta Informatica* 42, 2-3 (2005), 191–225.

[27] Simon J. Gay. 2016. Subtyping Supports Safe Session Substitution. In *A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*. 95–108.

[28] J.-Y. Girard. 1987. Linear Logic. *Theoret. Comput. Sci.* 50, 1 (1987), 1–102.

[29] Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. 1998. Language Primitives and Type Disciplines for Structured Communication-based Programming. In *ESOP'98 (LNCS)*, Vol. 1381. Springer-Verlag, 22–138.

[30] Kohei Honda, Nobuko Yoshida, and Marco Carbone. 2008. Multiparty asynchronous session types. In *POPL'08*. 273–284.

[31] Kohei Honda, Nobuko Yoshida, and Marco Carbone. 2016. Multiparty Asynchronous Session Types. *JACM* 63 (2016), 1–67. Issue 1-9.

[32] Raymond Hu and Nobuko Yoshida. 2016. Hybrid Session Verification Through Endpoint API Generation. In *FASE '16 (LNCS)*, Vol. 9633. Springer, 401–418.

[33] Raymond Hu and Nobuko Yoshida. 2017. Explicit Connection Actions in Multiparty Session Types. In *20th International Conference on Fundamental Approaches to Software Engineering (LNCS)*, Vol. 10202. Springer, 116–133.

[34] Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres Vieira, and Gianluigi Zavattaro. 2016. Foundations of Session Types and Behavioural Contracts. *ACM Comput. Surv.* 49, 1, Article 3 (April 2016), 36 pages.

[35] Limin Jia, Hannah Gommerstadt, and Frank Pfenning. 2016. Monitors and blame assignment for higher-order session types. In *POPL'16*. 582–594.

[36] Julien Lange and Emilio Tuosto. 2012. Synthesising Choreographies from Local Session Types. In *CONCUR'12 (LNCS)*, Maciej Koutny and Irek Ulidowski (Eds.), Vol. 7454. Springer, 225–239.

[37] Julien Lange, Emilio Tuosto, and Nobuko Yoshida. 2015. From Communicating Machines to Graphical Choreographies. In *POPL'15*, Sriram K. Rajamani and David Walker (Eds.). ACM Press, 221–232.

[38] Julien Lange and Nobuko Yoshida. 2016. Characteristic Formulae for Session Types. In *TACAS 2016 (LNCS)*, Vol. 9636. Springer, 833–850.

[39] Sam Lindley and J. Garrett Morris. 2015. A Semantics for Propositions as Sessions. In *ESOP'15 (LNCS)*, Vol. 9032. Springer, 560–584.

[40] Sam Lindley and J. Garrett Morris. 2016. Talking bananas: structural recursion for session types. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*. 434–447.

[41] Hugo A. Lopez, Eduardo R. B. Marques, Francisco Martins, Nicholas Ng, Casar Santos, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. 2015. Protocol-Based Verification of Message-Passing Parallel Programs. In *OOPSLA '15*. ACM, 280–298.

[42] Mungo 2016. Mungo homepage. http://www.dcs.gla.ac.uk/research/mungo/.

[43] Rumyana Neykova, Laura Bocchi, and Nobuko Yoshida. 2017. Timed runtime monitoring for multiparty conversations. *Formal Asp. Comput.* 29, 5 (2017), 877–910.

[44] Rumyana Neykova, Raymond Hu, Nobuko Yoshida, and Fahd Abdeljallal. 2018. Session Type Provider: Compile-time API Generation for Distributed Protocols in F#. In *27th International Conference on Compiler Construction*. ACM. To appear.

[45] Rumyana Neykova and Nobuko Yoshida. 2017. Let It Recover: Multiparty Protocol-Induced Recovery. In *26th International Conference on Compiler Construction*. ACM, 98–108.

[46] Rumyana Neykova and Nobuko Yoshida. 2017. Multiparty Session Actors. *LMCS* 13, 1 (2017).

[47] Nicholas Ng, Jose Coutinho, and Nobuko Yoshida. 2015. Protocols by Default: Safe MPI Code Generation based on Session Types. In *CC '15 (LNCS)*, Vol. 9031. Springer, 212–232.

[48] Nicholas Ng and Nobuko Yoshida. 2016. Static Deadlock Detection for Concurrent Go by Global Session Graph Synthesis. In *25th International Conference on Compiler Construction*. ACM, 174–184.

[49] Nicholas Ng, Nobuko Yoshida, and Kohei Honda. 2012. Multiparty Session C: Safe Parallel Programming with Message Optimisation. In *TOOLS '12 (LNCS)*, Vol. 7304. Springer, 202–218.

[50] Luca Padovani, Vasco Thudichum Vasconcelos, and Hugo Torres Vieira. 2014. Typing Liveness in Multiparty Communicating Systems. In *COORDINATION 2014 (LNCS)*, Vol. 8459. Springer, 147–162.
[51] Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. 2012. Linear Logical Relations for Session-Based Concurrency. In *ESOP'12 (LNCS)*, Vol. 7211. Springer, 539–558.
[52] Kirstin Peters, Uwe Nestmann, and Ursula Goltz. 2013. On Distributability in Process Calculi. In *ESOP (LNCS)*, Vol. 7792. Springer, 310–329.
[53] D. Sangiorgi. 1996. Pi-Calculus, Internal Mobility, and Agent Passing Calculi. *Theoretical Computer Science* 167, 1&2 (1996), 235–274.
[54] Alceste Scalas, Ornela Dardha, Raymond Hu, and Nobuko Yoshida. 2017. A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming. In *31st European Conference on Object-Oriented Programming, ECOOP 2017, June 19-23, 2017, Barcelona, Spain (LIPIcs)*, Vol. 74. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 24:1–24:31.
[55] Scribble. 2008. Scribble Project. www.scribble.org.
[56] K. C. Sivaramakrishnan, Mohammad Qudeisat, Lukasz Ziarek, Karthik Nagaraj, and Patrick Eugster. 2013. Efficient sessions. *Sci. Comput. Program.* 78, 2 (2013), 147–167.
[57] Ramsay Taylor, Emilio Tuosto, Neil Walkinshaw, and John Derrick. 2016. Choreography-Based Analysis of Distributed Message Passing Programs. In *PDP*. IEEE Computer Society, 512–519.
[58] Bernardo Toninho, Luís Caires, and Frank Pfenning. 2013. Higher-Order Processes, Functions, and Sessions: A Monadic Integration. In *ESOP'13 (LNCS)*, Vol. 7792. Springer, 350–369.
[59] Hugo Torres Vieira and Vasco Thudichum Vasconcelos. 2013. Typing Progress in Communication-Centred Systems. In *COORDINATION 2013 (LNCS)*, Vol. 7890. Springer, 236–250.
[60] Philip Wadler. 2014. Propositions as sessions. *J. Funct. Program.* 24, 2-3 (2014), 384–418.

A APPENDIX – PROOFS

A.1 Proofs from § 3 – Relating the CLL and MP systems

A.1.1 Proof of Proposition 3.7.

Proposition 3.7 (Operational Correspondence). *Suppose $P \Vdash^{\sigma}_{\rho} \Delta;\Gamma$ and $P \to P'$. Then $\rho(\sigma(P)) \to Q$ s.t. $P' \Vdash^{\sigma'}_{\rho'} \Delta';\Gamma'$ and $Q = \rho'(\sigma'(P'))$ with $\sigma' \subseteq \sigma$, $\rho' \subseteq \rho$.*

Proof. By induction on the given derivation. Since $\rho(\sigma(P)) \to Q$, $P \Vdash^{\sigma}_{\rho} \Delta;\Gamma$ is derived by applying (comp). Assume the last applied rule is

$$\frac{(\text{comp}) \quad P_1 \vdash^{\sigma}_{\mathsf{CL}} \Delta, x{:}A \quad P_2 \Vdash^{\sigma'}_{\rho} \Delta', x{:}A^{\perp};\Gamma \quad (\star)}{(\boldsymbol{\nu} x)(P_1 \mid P_2) \Vdash^{(\sigma'\cup\sigma)\setminus\{x\}}_{\rho'} \Delta, \Delta';\Gamma, \mathsf{c}_{\sigma}[\mathsf{p}_{\sigma}]{:}[\![P_1]\!]_{\sigma}}$$

and $(\boldsymbol{\nu} x)(P_1 \mid P_2) \to R$. Consider the case where $\to$ occurs due to a synchronisation on channel x, hence $R \equiv (\boldsymbol{\nu} x)(P_1' \mid P_2')$ with $P_1' \vdash^{\sigma_1}_{\mathsf{CL}} \Delta, x{:}A'$ and $P_2' \Vdash^{\sigma_2}_{\rho_2} \Delta', x{:}A'^{\perp};\Gamma'$ where $\sigma_1 \subseteq \sigma$, $\sigma_2 \subseteq \sigma'$, $\Gamma' \subseteq \Gamma$ and $\rho_2 \subseteq \rho \subseteq \rho'$. Hence by weakening the renamings appropriately and applying structural congruence, we have

$$P_1' \mid P_2' \Vdash^{(\sigma_1\cup\sigma_2)}_{\rho_2} \Delta, \Delta';\Gamma', \mathsf{c}_{\sigma_1}[\mathsf{p}_{\sigma_1}]{:}[\![P_1']\!]_{\sigma_1}$$

with $\rho(\sigma((\boldsymbol{\nu} x)(P_1 \mid P_2))) \to \rho(\sigma((\boldsymbol{\nu} x)(P_1' \mid P_2')))$ as required. The cases where either P_1 or P_2 reduce follow by i.h. □

Proposition A.1 (Renaming). *Suppose $P \Vdash^{\sigma}_{\rho} \Delta;\Gamma$. Then:*

(1) *for all bijective renamings φ on roles and channels, we have $P \Vdash^{\sigma\circ\varphi}_{\rho\circ\varphi} \Delta;\varphi(\Gamma)$.*
(2) *Assume $P \Vdash^{\sigma'}_{\rho'} \Delta;\Gamma'$. Then there exists a bijective renaming φ on roles and channels such that $\sigma' = \sigma \circ \varphi$ and $\rho' = \rho \circ \varphi$.*

Proof. By the definition of the mapping. □

Proof of Proposition 3.8.

Proposition 3.8 (Thread Preservation). *If $P \Vdash^{\sigma}_{\rho} \Delta;\Gamma$, then $\rho(\sigma(P))$ is thread-preserving.*

Proof. By induction on $P \Vdash^{\sigma}_{\rho} \Delta;\Gamma$. The case (thread) is by Definition 3.1. For the case (comp), by assumption $Q \Vdash^{\sigma'}_{\rho} \Delta', x{:}A^{\perp};\Gamma$, $\rho(\sigma'(Q))$ is thread-preserving. By the definition of (comp), we note that P is cut-free. Then by **(a)** in **(⋆)** in (comp), P's principal participant is p_σ. By Definition 3.1 and **(b,c)** in **(⋆)** in (comp), p_σ is disjoint with any principal participant in $\rho(\sigma'(Q))$. Hence the resulting process $\rho'((\sigma \cup \sigma \setminus \{x\})((\nu x)(P \mid Q)))$ is thread-preserving. □

Proof of Theorem 3.9.

Theorem 3.9 (Uniqueness). *Assume $P \vdash_{\mathsf{CL}} \Delta$. Suppose $\varphi(P)$ is thread-preserving and $\varphi(P)$ is typable by a single* MP *session, i.e. if $\varphi(P) \vdash_{\mathsf{MP}} \Gamma$ then* (1) $\mathrm{dom}(\Gamma)$ *contains a single session channel; or* (2) $\Gamma = \emptyset$ *and $P \equiv \mathbf{0}$. Then there exist ρ and σ such that $\varphi = \sigma \circ \rho$ and $P \Vdash^{\sigma}_{\rho} \Delta;\Gamma$.*

Proof. By induction on $P \vdash_{\mathsf{CL}} \Delta$. The case $P = \mathbf{0}$ and the case where P is a cut-free process are matched with (thread). By the assumption that a mapping is thread-preserving and Proposition A.1, we have that a cut-free process is mapped by (thread) with fixed σ (since (thread) is the only possible rule to create cuts). We show that if σ and ρ are fixed and a process is mapped into a single multiparty session, we must preserve the conditions in (⋆) of rule (comp).

The condition **(b)** of (⋆) must hold for P and Q can communicate. The condition $\mathsf{c}_\sigma = \mathsf{c}_{\sigma'}$ ensures a process is mapped into a single session. The condition $\mathsf{d}_\sigma(z), \mathsf{d}_{\sigma'}(y) \notin \rho$ in **(c)** is required to avoid a crash of the bound and free names. Finally $\mathsf{d}_\sigma(z) \neq \mathsf{d}_{\sigma'}(y)$ in **(c)** is required to map names in Δ and Δ' are mapped to different destinations (participants) to avoid a crash between the originally distinct names in CLL. Notice that without this condition, a parallel composition is untypable in MP since the same role indexed name $s[\mathsf{p}][\mathsf{q}]$ is spread into two parallel processes. □

A.2 Proofs from § 4 – Partial Multiparty Compatibility

A.2.1 Proof of Theorem 4.9.

Theorem 4.9 (Deadlock-freedom, MC and existence of a global type). *The following are equivalent:* (MC) *A configuration C is SMC;* (DF) *C is deadlock-free;* (WF) *There exists well-formed G such that $Tr(G) = Tr(C)$.*

Proof. **(DF)⇒(MC):** (a) Suppose configuration C_0 is deadlock-free. Then by DF, for all $C_0 \xrightarrow{\vec{\ell}} C$, there exists $C \xrightarrow{\ell_1 \cdot \overline{\ell_1}} C_1 \cdots C_n \xrightarrow{\ell_n \cdot \overline{\ell_n}} C'$ such that C' contains only end.

The base case $C_n = C_0 = C$ is obvious. Suppose $T_{1\mathsf{r}} \xrightarrow{\ell'_k} T_{k+1\mathsf{r}}$ and we are at C_k. In the case ℓ'_k is an output $\ell'_k = \mathsf{rq}{\uparrow}(S)$ or a selection $\ell'_k = \mathsf{rq} \triangleleft l$, there are traces such that $C_k \xrightarrow{\vec{\ell}'} C'_k$ which does not include the action from/to the participant p. Hence at C'_k, we have $T_{1\mathsf{r}} \xrightarrow{\ell'_k} T_{k+1\mathsf{r}}$ and $C'_k \xrightarrow{\ell'_k \cdot \overline{\ell'_k}} C''_k$. This matches with Definition 4.7(1). The case of the input $\ell'_k = \mathsf{rq}{\downarrow}(S)$ is similar, and matches with Definition 4.7(2). In the case ℓ'_k is a branching such that $\ell'_k = \mathsf{rq} \triangleright l$, we can reach C' which only contains end if and only if there exists $\ell''_k = \mathsf{rq} \triangleright l'$ such that $T_{1\mathsf{r}} \xrightarrow{\ell''_k} T'_{k+1\mathsf{r}}$ and $C_k \xrightarrow{\vec{\ell}'} C'_k \xrightarrow{\overline{\ell''_k} \cdot \ell''_k} C''_k$. This matches with Definition 4.7(3).

(MC)⇒(WF) By the synthesis theorem in [22].

(WF)⇒(DF) The trace of well-formed global types is DF by the definition of LTS of the global type G. Then by Proposition 4.6, C is DF.

□

A.2.2 *Proof of Lemma 4.14.*

Lemma 4.14. *Suppose* $\mathsf{fuse}(\mathsf{fuse}(G_i, G_j), G_k)$ *with* $\{i, j, k\} = \{1, 2, 3\}$ *is well-formed. Then we have* $\mathsf{fuse}(\mathsf{fuse}(G_i, G_j), G_k) \sim_{\mathsf{sw}} \mathsf{fuse}(G_i, \mathsf{fuse}(G_j, G_k))$.

Proof. Suppose $\mathsf{fuse}(\mathsf{fuse}(G_1, G_2), G_3)$ is defined. We proceed by the induction of the last rule applied to $\mathsf{fuse}(G_1, G_2)$. The only interesting cases are the first and second rules in Definition 4.10.

Case (1) Let $G_1 = \mathsf{p} \rightsquigarrow \mathsf{q}\!: \uparrow (T_1).G_1'$ and $G_2 = \mathsf{p} \rightsquigarrow \mathsf{q}\!: \downarrow (T_2).G_2'$ with $T_1 \geq T_2$. Then by the first fuse rule, we have

$$\mathsf{fuse}(G_1, G_2) = \mathsf{p} \to \mathsf{q}{:}(T_2).\mathsf{fuse}(G_1', G_2') = G_3'$$

Since $\mathsf{fuse}(\mathsf{fuse}(G_1, G_2), G_3)$ is defined, $\mathsf{p} \leftrightarrow \mathsf{q} \notin G_3$. By applying the third rule,

$$\mathsf{fuse}(G_3', G_3) = \mathsf{p} \to \mathsf{q}{:}(T_2).\mathsf{fuse}(\mathsf{fuse}(G_1', G_2'), G_3)$$

We now calculate $\mathsf{fuse}(G_2, G_3)$. Since $\mathsf{p} \leftrightarrow \mathsf{q} \notin G_3$, by applying the third rule, we have

$$\mathsf{fuse}(G_2, G_3) = \mathsf{p} \rightsquigarrow \mathsf{q}\!: \downarrow (T_2).\mathsf{fuse}(G_2', G_3) = G_3''$$

Then by applying the first rule, we have

$$\mathsf{fuse}(G_1, G_3'') = \mathsf{p} \to \mathsf{q}{:}(T_2).\mathsf{fuse}(G_1', \mathsf{fuse}(G_2', G_3))$$

By induction, $\mathsf{fuse}(\mathsf{fuse}(G_1', G_2'), G_3) \sim \mathsf{fuse}(G_1', \mathsf{fuse}(G_2', G_3))$. Hence we have $\mathsf{fuse}(\mathsf{fuse}(G_1, G_2), G_3) \sim \mathsf{fuse}(G_1, \mathsf{fuse}(G_2, G_3))$.

Case (2) The case $G_1 = \mathsf{p} \rightsquigarrow \mathsf{q}{:}\oplus\{l : G_1'\}$ and $G_2 = \mathsf{p} \rightsquigarrow \mathsf{q}\!: \&\, \{l : G_2', \{l_j : G_j\}_{j \in J}\}$ is similar to Case (1). Then by the second fuse rule, we have

$$\mathsf{fuse}(G_1, G_2) = \mathsf{p} \to \mathsf{q}{:}\{l : \mathsf{fuse}(G_1', G_2')\} = G_3'$$

Since $\mathsf{fuse}(\mathsf{fuse}(G_1, G_2), G_3)$ is defined, $\mathsf{p} \leftrightarrow \mathsf{q} \notin G_3$. By applying the forth rule,

$$\mathsf{fuse}(G_3', G_3) = \mathsf{p} \to \mathsf{q}{:}\{l : \mathsf{fuse}(\mathsf{fuse}(G_1', G_2'), G_3)\} = G_3''$$

We now calculate $\mathsf{fuse}(G_2, G_3)$. Since $\mathsf{p} \leftrightarrow \mathsf{q} \notin G_3$, by applying the forth rule, we have

$$\mathsf{fuse}(G_2, G_3) = \mathsf{p} \rightsquigarrow \mathsf{q}\!: \&\, \{l : \mathsf{fuse}(G_2', G_3), \{l_j : \mathsf{fuse}(G_j, G_3)\}_{j \in J}\} = G_3''$$

where the well-formedness guarantees $\mathsf{fuse}(G_2, G_3)$ is defined. Then by applying the second rule, we have

$$\mathsf{fuse}(G_1, G_3'') = \mathsf{p} \to \mathsf{q}{:}\{l : \mathsf{fuse}(G_1, \mathsf{fuse}(G_2', G_3))\}$$

By induction, $\mathsf{fuse}(\mathsf{fuse}(G_1', G_2'), G_3) \sim \mathsf{fuse}(G_1', \mathsf{fuse}(G_2', G_3))$. Hence we have $\mathsf{fuse}(\mathsf{fuse}(G_1, G_2), G_3) \sim \mathsf{fuse}(G_1, \mathsf{fuse}(G_2, G_3))$.

□

A.2.3 *Proof of Theorem 4.15.*

Theorem 4.15 (Compositionality). *Suppose* $G_1, \ldots, G_n$ *are partial global types. Assume* $\forall i, j$ *such that* $1 \leq i \neq j \leq n$, G_i *and* G_j *are PMC and* $G = \mathsf{fuse}(G_1, \mathsf{fuse}(G_2, \mathsf{fuse}(\ldots, G_n)))$ *is a complete global type. Then* G *is well-formed.*

Proof. By Lemma 4.14, we need only show the case where $G_{n-1}' = \mathsf{fuse}(\cdots(\mathsf{fuse}(G_1, G_2), G_3), \cdots, G_{n-1})$ and $\mathsf{fuse}(G_{n-1}', G_n)$ is complete. We proceed by induction on n.

Case $n = 2$ Suppose there are two participants p or q and G_1 contains $\mathsf{p}_1 \rightsquigarrow \mathsf{p}_1' \cdots \mathsf{p}_m \rightsquigarrow \mathsf{p}_m'$. Then $\mathsf{p}_i = \mathsf{p} \wedge \mathsf{p}_i' = \mathsf{q}$ or $\mathsf{p}_i = \mathsf{q} \wedge \mathsf{p}_i' = \mathsf{p}$. Note that we cannot apply the swapping relation between $\mathsf{p}_i \rightsquigarrow \mathsf{p}_i'$ and $\mathsf{p}_j \rightsquigarrow \mathsf{p}_j'$ since all actions have the same principal name ($\mathsf{pr}(G_1) = \{\mathsf{p}\}$ and $\mathsf{pr}(G_2) = \{\mathsf{q}\}$). Then G_2 should contain $\mathsf{p}_1 \rightsquigarrow \mathsf{p}_1' \cdots \mathsf{p}_m \rightsquigarrow \mathsf{p}_m'$ with dual modes since the last five rules are not applicable by the side conditions $\mathsf{p} \leftrightarrow \mathsf{q} \notin G_2$.

Suppose G'_{n-1} contains n participants and there are only partial arrows from p_n or to p_n and $\mathrm{fuse}(G'_{n-1}, G_n)$ is complete. Then the partial arrows in G'_{n-1} form (possibly more than one) chains such that $\mathsf{p}_1 \rightsquigarrow \mathsf{p}'_1 \cdots \mathsf{p}_m \rightsquigarrow \mathsf{p}'_m$ where either p_i or p'_i is p_n. To obtain a complete global type, we must have dual chains $\mathsf{p}_1 \rightsquigarrow \mathsf{p}'_1 \cdots \mathsf{p}_m \rightsquigarrow \mathsf{p}'_m$ in G_n. Assuming the completed $n-1$ participants in G'_{n-1} form a well-formed global type, applying fuse rules one by one from the head (note that the last five rules are not applicable for the partial arrow $\mathsf{p}_i \rightsquigarrow \mathsf{p}'_i$ in G_1 by the side condition $\mathsf{p}_i \leftrightarrow \mathsf{p}'_i \notin G_n$), we see that $\mathrm{fuse}(G'_{n-1}, G_n)$ is well-formed. □

A.3 Proofs from § 5 – CLL encoded as a single multiparty session

A.3.1 *Proof of Lemma 5.8.*

Lemma 5.8. *Let* $P \Vdash^{\sigma}_{\rho} \Delta; \Gamma; \mathcal{G}$. $\mathrm{co}(\Gamma)$ *implies* $\Delta = \emptyset$ *or* Δ *contains only* $\mathbf{1}$ *or* $\perp$ *and* $\sigma = \emptyset$.

Proof. Assume to the contrary that $\Delta \neq \emptyset$ and does not contain only $\mathbf{1}$ and $\perp$, or $\sigma \neq \emptyset$. Then it must be the case that P has some free name $x{:}A \in \Delta$ where $\sigma(x) = s[\mathsf{p}][\mathsf{q}]$, for some $s, \mathsf{p}, \mathsf{q}$ with $A \neq \mathbf{1}$ or $\perp$. By construction it must necessarily be the case that $s[\mathsf{p}]{:}T \in \Gamma$. Since x is free in P, we cannot have $s[\mathsf{q}]{:}T' \in \Gamma$ with $T{\upharpoonright}\mathsf{q} \leq \overline{T'{\upharpoonright}\mathsf{p}}$: single thread σ-renamings are invariant on the p role name, so for $s[\mathsf{q}]$ to occur in Γ it must have arose due to composition on x, which is impossible since x is free, or on some other (now) bound name y that mapped to $s[\mathsf{q}][\mathsf{r}]$, for some r. However, since $\sigma(x) = s[\mathsf{p}][\mathsf{q}]$, by construction we know that $\mathsf{q} \notin \rho$. This is contradictory with the assumption of coherence and so we conclude the proof. □

A.3.2 *Proof of Lemma 5.9.*

Lemma 5.9. *Let* $P \Vdash^{\emptyset}_{\rho} \Delta; \Gamma; \mathcal{G}$ *with* $\Delta = \emptyset$ *or* Δ *containing only* $\mathbf{1}$ *or* $\perp$. *We have that* $\mathrm{co}(\Gamma)$.

Proof. Assume to the contrary that Γ is not coherent. Then (1) there exists $s[\mathsf{p}]{:}T, s[\mathsf{q}]{:}T'$ in Γ such that $T{\upharpoonright}\mathsf{q} \not\leq \overline{T'{\upharpoonright}\mathsf{p}}$ or (2) $s[\mathsf{p}] : T \in \Gamma$ such that $\mathsf{q} \in \mathrm{roles}(T)$ and $s[\mathsf{q}] : T' \notin \Gamma$.

For (1) to be the case, either $T{\upharpoonright}\mathsf{q}$ contains an action unmatched by $T'{\upharpoonright}\mathsf{p}$ or vice-versa. Assume wlog that $T{\upharpoonright}\mathsf{q}$ contains an unmatched action. Since both $s[\mathsf{p}]{:}T \in \Gamma$ and $s[\mathsf{q}]{:}T' \in \Gamma$ we know that there exists $(x, s[\mathsf{p}][\mathsf{q}]) \in \rho$ where some subprocess of P uses $x{:}A$ for some A and some other subprocess of P uses $\overline{x}{:}A^{\perp}$. The duality of $x{:}A$ and $x{:}A^{\perp}$ contradicts the existence of an unmatched action in $T{\upharpoonright}\mathsf{q}$. The argument for an unmatched action in $T'{\upharpoonright}\mathsf{p}$ is identical.

For (2) to be the case, since $s[\mathsf{p}]{:}T \in \Gamma$ and $s[\mathsf{q}]{:}T' \notin \Gamma$, we know that there exists $(x, s[\mathsf{p}][\mathsf{r}]) \in \rho$, with $\mathsf{r} \neq \mathsf{q}$, and that $(z, s[\mathsf{q}][\mathsf{s}]) \notin \rho$. Since $q \in \mathrm{roles}(T)$ then it must be the case that P uses a channel y mapped to $s[\mathsf{p}][\mathsf{q}]$ that is free, which contradicts our assumptions. □

A.4 Proofs from § 6 – Higher-Order Channel Passing

A.4.1 *Proof of Theorem 6.10.*

Theorem 6.10 (Consistency). *Assume* $P \vdash_{\mathsf{CL}} \Delta$. *If* $\varphi(P)$ *is typable by* Γ, *i.e.* $\varphi(P) \vdash_{\mathsf{MP}} \Gamma$ *and* ρ, σ *and* η *satisfy the conditions in Definitions 6.1 and 6.2 and* $\varphi = \sigma \circ \eta \circ \rho$, *then* $P \vDash^{\sigma,\eta}_{\rho} \Delta; \Gamma; \mathcal{G}$.

Proof. By induction on $P \vdash^{\sigma,\eta}_{\mathsf{CL}} \Delta$ (note that $\mathcal{G}$ does not affect the proof, hence we omit). Given bijective renaming $\varphi = \sigma \circ \eta \circ \rho$ which satisfies Definition 6.1 and Definition 6.2, we prove the conditions in (comp_d) are ensured by typability of MP under some σ, η and ρ which satisfy Definition 6.1 and Definition 6.2. The case for conditions **(a,b,c)** is proved as Theorem 6.10. Condition **(d)** is ensured by the conditions of φ; and condition **(e)** is satisfied by the assumption that a delegated bound name coincides with a input variable in the receiver side; and **(d)** is ensured by a disjointness of free names and bound names in typable MP. □

A.4.2 *Proof of Lemma 6.13.*

Lemma 6.13. *Let* $P \vDash_{\rho}^{\emptyset,\eta} \Delta; \Gamma; \mathcal{G}$. $\mathrm{co}(\Gamma)$ *implies* $\Delta = \emptyset$ *or* Δ *containing only* $\mathbf{1}$ *or* $\bot$.

Proof. We note that the renamings ensure that the two endpoints of an interaction cannot be implemented by the same single-thread process and that bound-names involved in delegation denote linear interactions along different session channels.

Assume to the contrary that $\Delta \neq \emptyset$ and does not contain only $\mathbf{1}$ and $\bot$, or $\sigma \neq \emptyset$. Then it must be the case that P has some free name $x{:}A \in \Delta$ where $\sigma(x) = s[\mathrm{p}][\mathrm{q}]$, for some $s, \mathrm{p}, \mathrm{q}$ with $A \neq \mathbf{1}$ or $\bot$. By construction it must necessarily be the case that $s[\mathrm{p}]{:}T_1 \in \Gamma$. However, since x is free in P we have that if $s[\mathrm{q}]{:}T_2 \in \Gamma$ then $T_1 \upharpoonright \mathrm{q} \not\leq T_2 \upharpoonright \mathrm{p}$. For T_2 to have the corresponding actions with role p there must exist a free name y in P such that $\sigma(y) = s[\mathrm{q}][\mathrm{p}]$. If both x and y are in the same cut-free sub-process this contradicts Definition 6.2. If they are in different sub-processes, this contradicts the premise of the composition rule. The only remaining possibility is for a bound name of P to have been mapped to $s[\mathrm{q}][\mathrm{p}]$, which also contradicts the premise of the composition rule. This arguments contradicts the assumption of coherence and so we conclude the proof. □