



# Multicompatibility for Multiparty-Session Composition

Franco Barbanera

University of Catania, Italy, barba@dm.unict.it

Lorenzo Gheri

University of Oxford, UK, lorenzo.gheri@cs.ox.ac.uk

Mariangiola Dezani-Ciancaglini

University of Torino, Italy, dezani@di.unito.it

Nobuko Yoshida

University of Oxford, UK, nobuko.yoshida@cs.ox.ac.uk

## ABSTRACT

Modular methodologies for the development and verification of concurrent/distributed systems are increasingly relevant nowadays. We investigate the simultaneous composition of multiple systems in a multiparty-session-type setting, working on suitable notions of *interfacing policy* and *multicompatibility*. The resulting method is conservative (it makes only the strictly needed changes), flexible (any system can be looked at as potentially open) and safe (relevant communication properties, e.g. *lock-freedom*, are preserved by composition). We obtain safety by proving preservation of typability. We also provide a sound and complete type inference algorithm.

## CCS CONCEPTS

• **Theory of computation** → **Process calculi**; **Type theory**.

## KEYWORDS

Multiparty Sessions, Global Types, Open System Composition, Modularity.

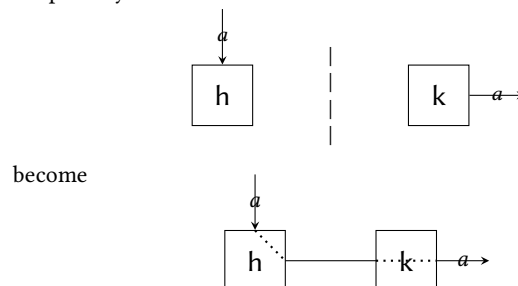
## ACM Reference Format:

Franco Barbanera, Mariangiola Dezani-Ciancaglini, Lorenzo Gheri, and Nobuko Yoshida. 2023. Multicompatibility for Multiparty-Session Composition. In *International Symposium on Principles and Practice of Declarative Programming (PPDP 2023)*, October 22–23, 2023, Lisboa, Portugal. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3610612.3610614>

## 1 INTRODUCTION

Verification of communication patterns is of central importance for concurrent/distributed implementations of multiple communicating participants, as well as the possibility of ensuring good behavioural properties (e.g., lock-freedom). The shortcoming of many approaches to such an issue, both structured, – e.g., MultiParty Session Types (MPSTs) [21, 22] – and unstructured – e.g. Communicating Finite State Machines (CFSMs) [9] – is to design and analyse communicating systems as *stand-alone closed entities*: the designer/analysers has *full knowledge* of every and each interaction between any two participants. This hinders modularity features, which are crucial for the specification and development of large-scale, complex, distributed communicating systems.

Realistically, systems should be open, i.e. liable to interact with an external environment (typically other systems). In [3] an approach to (binary) composition – dubbed *Participants-as-Interfaces* (PaI) *composition* – was devised enabling to look at any system, even closed ones, as virtually open. In a nutshell, given two systems, one first selects two participants – one per system – which exhibit “compatible behaviours”; then transforms them into coupled gateways connecting the two systems. Such gateways work simply as “forwarders”: a message intended for the interface-participant in one system is instead received by the gateway and immediately forwarded to the coupled gateway in the other system which, in turn, sends it to appropriate participants. For example, if one interface is ready to receive a message and another interface is ready to send the same message, then the gateway replacing the first interface will forward the received message to the gateway replacing the second interface. Essentially the gateways are obtained from the interfaces by adding forwarding of messages between them. Graphically



This composition mechanism is “conservative”, i.e. it makes only the strictly needed changes; and “flexible”, i.e., it allows to look at any system as potentially open. The PaI approach was exploited in a number of papers for both MPSTs [4, 5] and CFSMs [3, 6, 8], where another essential feature of this approach was proved: *safety*. Safe composition mechanisms being those that do not “break” any relevant property of the single systems. A drawback of the above mentioned investigations on PaI is that they have been carried out for binary composition only; or for a restricted notion of multiple connection in a client-server setting [4].

In the present paper, we push forward the PaI composition, exploring the setting of *multiple* simultaneous composition of several sessions.<sup>1</sup> Note that if we compose, two by two, several sessions using binary composition, we get to tree-like structures only. In fact, by looking at sessions as vertices and gateway connections as undirected edges, the only way to get a cycle using binary composition is by connecting two interfaces belonging to the same composed session.



This work is licensed under a Creative Commons Attribution International 4.0 License.

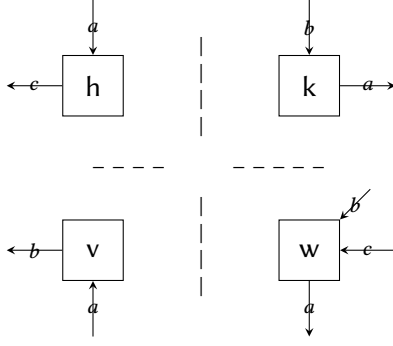
PPDP 2023, October 22–23, 2023, Lisboa, Portugal

© 2023 Copyright held by the owner/author(s).

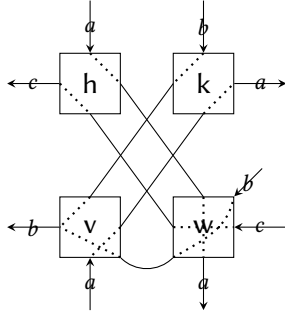
ACM ISBN 979-8-4007-0812-1/23/10.

<https://doi.org/10.1145/3610612.3610614>

<sup>1</sup>A (multiparty) session, i.e. a set of named processes in parallel, formalises in this paper the general notion of “system”.



In order to illustrate the idea underlying *PaI multicomposition* and its related issues, let us assume to have four sessions,  $\mathbb{M}_1$ ,  $\mathbb{M}_2$ ,  $\mathbb{M}_3$  and  $\mathbb{M}_4$ , containing respectively four participants,  $h$ ,  $k$ ,  $v$  and  $w$ , that we decide to transform into gateways (if possible) enabling to connect the four sessions into a single one. For the sake of simplicity we abstract here from the way communications are performed and from the logical order of the exchanged messages. The drawing above represents the messages the participants do exchange inside their respective sessions. The composition of the four sessions then consists in replacing the participants  $h$ ,  $k$ ,  $v$  and  $w$ , chosen as interfaces, by gateways. Note that a message, say the  $a$  that in  $\mathbb{M}_1$  is sent to  $h$ , could be forwarded (unlike the binary case) to different other gateways. This means that an *interfacing policy* has to be set up in order to appropriately define the gateways. An interfacing policy for the present example could be for instance the one that forwards to  $w$  the  $a$  received by  $h$ ; to  $k$  the  $a$  received by  $v$ ; to  $v$  the  $b$ 's received by  $w$  and  $k$ ; to  $h$  the  $c$  received by  $w$ . According to such an interfacing policy, the interface participants are replaced by gateways as described below.



We notice that such a composition cannot be done using the mechanisms currently available in the literature [3–6, 8].

The key issue when implementing such a simple idea into a particular formalism is to ensure the above construction to be *safe*. In the binary case safety can be ensured by the duality of the interfaces [3, 5, 6, 8]. For client-server composition, the compliance of server interfaces with the client interface allows to obtain safe compositions [4]. Here the situation is more tricky; we shall preserve the system properties by identifying a correct notion of compatibility for *PaI multicomposition* (which we dub *multicompatibility*) in the setting of MPSTs.

In MPST approaches to session specification and verification, two phases are distinguished: *implementation* (where code is independently and distributively written for each single participant) and *verification*. The former can happen after the latter, as an

analysis of existing code; or before, with a sound design of the communicating systems (and, e.g., the generation of APIs that will guide the programmers). MPSTs have spawned a variety of tools for the sound modular implementation of a communicating system [13, 20, 27, 29, 32, 38, 39]: the code is safe, as long as it is well typed. However, in the vast majority of the MPST literature [14, 18, 21, 22, 32, 37], the verification of the whole session is treated as a single, centralised operation. More precisely the behaviours of sessions are described by *global types* prescribing the order and the type of the communications between session participants. The MPST literature dealing with modular verification and composition of open sessions is however still at its early stages [4, 5, 19, 36]. In particular the present paper is the first one dealing with the safe *PaI* composition of an arbitrary number of sessions.

**Contributions and Structure of the Paper.** This paper introduces a *conservative*, *flexible* and *safe* *PaI* multicomposition method based on the notions of interfacing policy and multicompatibility, thus improving on the state of the art [4, 5, 19, 36]. In Section 2 we recall the MPST calculus of multiparty sessions with its type system, as defined in [4]: we note that well-typed sessions are lock-free. Section 3 contains our main contributions. In particular, a precise notion of interfacing policy (Definition 3.6) is identified for an arbitrary number of multiparty sessions; building on that, multicompatibility (Definition 3.8) is defined in terms of typability of any of the possible interfacing policies. *PaI* multicomposition for sessions is then given in terms of interfacing policies (Definition 3.12). It is *safe* since we prove that multicomposition of multicompatible sessions is typable (Theorem 3.17), and hence lock-free. In Section 4 we define an *inference algorithm* for the global types of an arbitrary session, if any. We prove the *soundness and completeness* of this algorithm (Theorem 4.7). Section 5 discusses related works and concludes the paper.

## 2 THE CALCULUS OF MULTIPARTY SESSIONS AND ITS TYPE SYSTEM

In the present section we recall the calculus of multiparty sessions and its type system as defined in [4], to which we refer for more detailed explanations and for proofs.

We assume to have the following denumerable base sets: *messages* (ranged over by  $\lambda, \lambda', \dots$ ); *session participants* (ranged over by  $h, p, q, r, \dots$ ); *indexes* (ranged over by  $i, j, l, n, \dots$ ); *sets of indexes* (ranged over by  $I, J, \dots$ ).

Processes, ranged over by  $P, Q, R, S, H, K, \dots$ , implement the behaviour of participants. In the following and in later definitions the symbol  $::=^{coind}$  will indicate that the productions have to be interpreted *coinductively* and that only *regular* terms are allowed. Then we can adopt in proofs the coinduction style advocated in [25] which, without any loss of formal rigour, promotes readability and conciseness.

Multiparty sessions are parallel compositions of pairs participant/process of the form  $p[P]$ .

**Definition 2.1 (Processes and Multiparty Sessions).** Processes are defined by:

$$P ::=^{coind} \mathbf{0} \mid p!\{\lambda_i.P_i\}_{i \in I} \mid p?\{\lambda_i.P_i\}_{i \in I}$$

$$[\text{COMM-T}] \frac{l \in I \subseteq J}{p[q!\{\lambda_i.P_i\}_{i \in I}] \parallel q[p?\{\lambda_j.Q_j\}_{j \in J}] \parallel \mathbb{M} \xrightarrow{p\lambda_l q} p[P_l] \parallel q[Q_l] \parallel \mathbb{M}}$$

Figure 1: LTS for multiparty sessions.

where  $I \neq \emptyset$  and  $\lambda_j \neq \lambda_l$  for  $j, l \in I$  and  $j \neq l$ .

*Multiparty sessions* (sessions, for short) are expressions of the shape:

$$p_1[P_1] \parallel \dots \parallel p_n[P_n]$$

where  $p_j \neq p_l$  for  $1 \leq j, l \leq n$  and  $j \neq l$ . We use  $\mathbb{M}$  to range over multiparty sessions.

In the above definition, the output process  $p!\{\lambda_i.P_i\}_{i \in I}$  non-deterministically chooses one message  $\lambda_i$  for some  $i \in I$ , and sends it to the participant  $p$ , thereafter continuing as  $P_i$ . Symmetrically, the input process  $p?\{\lambda_i.P_i\}_{i \in I}$  waits for one of the messages  $\lambda_i$  from the participant  $p$ , then continues as  $P_i$  after receiving it. When there is only one output we write  $p!\lambda.P$  and similarly for one input. We use  $0$  to denote the terminated process.

We assume the standard structural congruence  $\equiv$  on multiparty sessions, stating that parallel composition is associative and commutative and has neutral elements  $p[0]$  for any  $p$ . If  $P \neq 0$  we write  $p[P] \in \mathbb{M}$  as short for  $\mathbb{M} \equiv p[P] \parallel \mathbb{M}'$  for some  $\mathbb{M}'$ . We shall also write  $\prod_{i=1}^n p_i[P_i]$  as short for  $p_1[P_1] \parallel \dots \parallel p_n[P_n]$ .

The *set of participants* of a session  $\mathbb{M}$ , notation  $\text{prt}(\mathbb{M})$ , is as expected:

$$\text{prt}(\mathbb{M}) = \{p \mid p[P] \in \mathbb{M}\}$$

To define the *synchronous operational semantics* of sessions we use an LTS, whose transitions are decorated by communications.

**Definition 2.2 (LTS for Multiparty Sessions).** The *labelled transition system (LTS) for multiparty sessions* is the closure under  $\equiv$  of the reduction specified by the unique rule shown in Figure 1.

Rule [COMM-T] makes the communication possible: participant  $p$  sends message  $\lambda_l$  to participant  $q$ . This rule is non-deterministic in the choice of messages. The condition  $I \subseteq J$  ensures that the sender can freely choose the message, since the receiver must offer all sender messages and possibly more. This allows us to distinguish in the operational semantics between internal (output) and external (input) choices. Note that this condition will always be true in well-typed sessions.

Communications are triples of the form  $p\lambda q$  ranged over by  $\Lambda, \Lambda', \dots$ . We define *traces* as (possibly infinite) sequences of communications by:

$$\sigma ::= \text{coind } \epsilon \mid \Lambda \cdot \sigma$$

where  $\epsilon$  is the empty sequence. We use  $|\sigma|$  to denote the length of the trace  $\sigma$ , where  $|\sigma| = \infty$  when  $\sigma$  is an infinite trace. We define the participants of communications and traces:

$$\text{prt}(p\lambda q) = \{p, q\} \quad \text{prt}(\epsilon) = \emptyset \quad \text{prt}(\Lambda \cdot \sigma) = \text{prt}(\Lambda) \cup \text{prt}(\sigma)$$

When  $\sigma = \Lambda_1 \cdot \dots \cdot \Lambda_n$  ( $n \geq 0$ ) we write  $\mathbb{M} \xrightarrow{\sigma} \mathbb{M}'$  as short for

$$\mathbb{M} \xrightarrow{\Lambda_1} \mathbb{M}_1 \dots \xrightarrow{\Lambda_n} \mathbb{M}_n = \mathbb{M}'$$

We give now a very simple example, that shall be used throughout the paper in order to clarify the notions we introduce.

*Example 2.3 (Working example).* Let us consider a session with two participants<sup>2</sup>:

$$\mathbb{M}_1 = h_1[H_1] \parallel p[P]$$

Process  $H_1$  controls the entrance of customers in a mall (via some sensor). As soon as a customer enters,  $H_1$  sends a message  $\text{start}$  to the process  $P$  which controls a display for advertisements. After the start message,  $P$  displays a general advertising image. Process  $P$  does control also a sensor detecting emotional reactions as well as a card reader distinguishing regular from new customers. Such information, through the messages  $\text{react}$ ,  $\text{rc}$  and  $\text{nc}$  is sent to  $H_1$ . Using that information  $H_1$  sends to  $P$  a customised image, depending on the kind of the customer, through message  $\text{img}$ . The processes of such a session can then be defined as follows

$$\begin{aligned} H_1 &= p!\text{start}. p?\text{react}. p? \left\{ \begin{array}{l} \text{rc}. p!\text{img}. H_1 \\ \text{nc}. p!\text{img}. H_1 \end{array} \right. \\ P &= h_1?\text{start}. h_1!\text{react}. h_1! \left\{ \begin{array}{l} \text{rc}. h_1?\text{img}. P \\ \text{nc}. h_1?\text{img}. P \end{array} \right. \end{aligned}$$

where sets of alternatives are denoted by branchings.  $\diamond$

Lock-freedom for multiparty sessions is defined as in [24, 30]. In words, each participant ready to communicate is never prevented from finding a partner exposing a dual communication action. Lock-freedom ensures progress for each participant, and hence deadlock-freedom.

**Definition 2.4 (Lock-freedom).** A multiparty session  $\mathbb{M}$  is a *lock-free session* if  $\mathbb{M} \xrightarrow{\sigma} \mathbb{M}'$  and  $p[P] \in \mathbb{M}'$  imply  $\mathbb{M}' \xrightarrow{\sigma' \cdot \Lambda} \mathbb{M}''$  for some  $\sigma'$  and  $\Lambda$  such that  $p \in \text{prt}(\Lambda)$ .

Notice that we need to consider  $\mathbb{M}'$  in the above definition, since otherwise the multiparty session  $p[q!\lambda.0] \parallel q[p?\lambda.p?\lambda.0]$  would be lock-free.

We recall now the type system of [4], in which sessions are directly typed by global types without using projections [21, 22]. If the global type respects a well-formedness condition (namely boundedness, see Definition 2.7), the typed session does evolve in agreement with what the global type prescribes (subject reduction and session fidelity) and lock-freedom is ensured.

**Definition 2.5 (Global Types).** *Global types* are defined by:

$$G ::= \text{coind } \text{End} \mid p \rightarrow q : \{\lambda_i.G_i\}_{i \in I}$$

where  $I \neq \emptyset$  and  $\lambda_j \neq \lambda_l$  for  $j, l \in I$  and  $j \neq l$ .

The type  $p \rightarrow q : \{\lambda_i.G_i\}_{i \in I}$  formalises a protocol where participant  $p$  must send to  $q$  a message  $\lambda_j$  for some  $j \in I$ , (and  $q$  must

<sup>2</sup>For the sake of simplicity, in our examples we consider only sessions with two or three participants. Our definitions and results are however independent from the number of participants in the single sessions.

$$\begin{array}{c}
\text{[End]} \frac{}{\text{End} \vdash p[0]} \quad \text{[COMM]} \frac{G_i \vdash p[P_i] \parallel q[Q_i] \parallel \mathbb{M} \quad \text{prt}(G_i) \setminus \{p, q\} = \text{prt}(\mathbb{M}) \quad \forall i \in I}{p \rightarrow q : \{\lambda_i.G_i\}_{i \in I} \vdash p[q!\{\lambda_i.P_i\}_{i \in I}] \parallel q[p?\{\lambda_j.Q_j\}_{j \in J}] \parallel \mathbb{M}} \quad I \subseteq J
\end{array}$$

Figure 2: Typing rules.

$$\begin{array}{c}
\mathcal{D} = \frac{\frac{\frac{h_1 \rightarrow p:\text{IMG}. G_1 \vdash h_1[p!\text{IMG}. H_1] \parallel p[h_1?\text{IMG}. P]}{\mathcal{D}} \quad \frac{\frac{h_1 \rightarrow p:\text{IMG}. G_1 \vdash h_1[p!\text{IMG}. H_1] \parallel p[h_1?\text{IMG}. P]}{\mathcal{D}}}{\frac{G'_1 \vdash h_1[H'_1] \parallel p[P_1]}{p \rightarrow h_1:\text{REACT}. G'_1 \vdash h_1[p?\text{REACT}. H'_1] \parallel p[h_1!\text{REACT}. P_1]}} \\
\frac{}{G_1 \vdash h_1[H_1] \parallel p[P]}
\end{array}$$

Figure 3: Derivation of Example 2.10.

receive it) and then, depending on which  $\lambda_j$  was chosen by  $p$ , the protocol continues as  $G_j$ . The notation  $p \rightarrow q : \lambda.G$  is used when there is only one message. The terminal symbol  $\text{End}$  denotes the terminated protocol.

The *set of paths of a global type*  $G$ , notation  $\text{paths}(G)$ , is defined as the greatest set such that:

$$\begin{aligned}
\text{paths}(\text{End}) &= \{\epsilon\} \\
\text{paths}(p \rightarrow q : \{\lambda_i.G_i\}_{i \in I}) &= \bigcup_{i \in I} \{p\lambda_i q \cdot \sigma \mid \sigma \in \text{paths}(G_i)\}
\end{aligned}$$

Clearly, paths of global types are traces as defined after Definition 2.2. The *set of participants of a global type* is the set of participants of its paths:

$$\text{prt}(G) = \bigcup_{\sigma \in \text{paths}(G)} \text{prt}(\sigma)$$

For any  $G$ , regularity of global types ensures  $\text{prt}(G)$  to be finite.

In order to ensure lock-freedom by typing, each participant is required to occur in all the paths from the root. Technically, this is obtained by means of the notions of *depth* and of *bounded* type below. The  $n$ -th communication in a path  $\sigma$ , where  $n \in \mathbb{N}$  and  $1 \leq n \leq |\sigma|$ , is denoted by  $\sigma[n]$ .

**Definition 2.6 (Depth).** Let  $G$  be a global type. For  $\sigma \in \text{paths}(G)$  we define

$$\text{depth}(\sigma, p) = \inf\{n \mid p \in \text{prt}(\sigma[n])\}$$

and define  $\text{depth}(G, p)$ , the *depth* of  $p$  in  $G$ , as follows:

$$\text{depth}(G, p) = \begin{cases} \sup\{\text{depth}(\sigma, p) \mid \sigma \in \text{paths}(G)\} & \text{if } p \in \text{prt}(G) \\ 0 & \text{otherwise} \end{cases}$$

**Definition 2.7 (Boundedness).** A global type  $G$  is *bounded* if  $\text{depth}(G', p)$  is finite for all participants  $p \in \text{prt}(G')$  and all types  $G'$  which occur in  $G$ .

Intuitively, this means that if  $p \in \text{prt}(G')$  for a subexpression of  $G$  which is a type, then the search for an interaction of the shape  $p\lambda q$  or  $q\lambda p$  along a path  $\sigma \in \text{paths}(G')$  terminates (and recall that  $G'$  can be infinite, in which case  $G$  is such). As shown in [4, Example 2], it is necessary to consider all types occurring in a global type when defining boundedness.

Since global types are regular, the boundedness condition is decidable. Only bounded global types will be allowed in typing sessions.

The simplicity of the multiparty session calculus allows to formulate a type system deriving directly global types for multiparty sessions, i.e. judgments of the form  $G \vdash \mathbb{M}$  (where  $G$  is bounded). Here and in the following, the double line indicates that the rules are interpreted coinductively [31, Chapter 21].

**Definition 2.8 (Type System).** The type system is defined by the axiom and rule in Figure 2, where sessions are considered modulo structural equivalence.

Rule [COMM] just adds simultaneous communications to global types and to corresponding processes inside sessions. Note that this rule allows more inputs than corresponding outputs, in agreement with the condition in Rule [COMM-T] (Definition 2.2). It also allows more branches in the input process than in the global type, just mimicking the subtyping for session types [17]. Instead, the number of branches in the output process and the global type must be the same. This does not restrict typability as shown in [5], while it improves session fidelity as discussed after Theorem 2.13. The condition  $\text{prt}(G_i) \setminus \{p, q\} = \text{prt}(\mathbb{M})$  for all  $i \in I$  ensures that the global type and the session have exactly the same set of participants. In this way we forbid for example to derive

$$p \rightarrow q : \lambda.\text{End} \vdash p[q!\lambda.0] \parallel q[p?\lambda.0] \parallel r[R] \text{ with } R \neq 0$$

arbitrary.

The regularity of processes and global types ensures the decidability of type checking. Besides, it is worth also remarking that typability alone does not ensure boundedness of types as shown in the following example.

**Example 2.9 (Typability does not ensure boundedness).** The following global type is unbounded, since  $\text{depth}(G', r) = \infty$ :

$$G = r \rightarrow q : \lambda.G' \text{ where } G' = p \rightarrow q : \{\lambda_1.q \rightarrow r : \lambda'.\text{End}, \lambda_2.G'\}$$

Without the boundedness condition we can assign  $G$  to the session  $p[P] \parallel q[r?\lambda.Q] \parallel r[q!\lambda.q?\lambda'.0]$ , where  $P = q!\{\lambda_1.0, \lambda_2.P\}$  and  $Q = p?\{\lambda_1.r!\lambda'.0, \lambda_2.Q\}$ .  $\diamond$

**Example 2.10 (Typing the multiparty session of Example 2.3).** It is easy to check that, for the multiparty session  $\mathbb{M}_1$  of Example 2.3, we can derive  $G_1 \vdash \mathbb{M}_1$  with the derivation  $\mathcal{D}$  of Figure 3, where

$$G_1 = h_1 \rightarrow p:\text{START}. p \rightarrow h_1:\text{REACT}. G'_1$$

$$\begin{array}{c}
\text{[ECOMM]} \quad \frac{j \in I}{p \rightarrow q : \{\lambda_i.G_i\}_{i \in I} \xrightarrow{p\lambda_j q} G_j} \\
\text{[ICOMM]} \quad \frac{G_i \xrightarrow{p\lambda q} G'_i \quad \forall i \in I \quad \{p, q\} \cap \{r, s\} = \emptyset}{r \rightarrow s : \{\lambda_i.G_i\}_{i \in I} \xrightarrow{p\lambda q} r \rightarrow s : \{\lambda_i.G'_i\}_{i \in I}}
\end{array}$$

Figure 4: LTS for global types.

$$\begin{aligned}
G'_1 &= p \rightarrow h_1 : \begin{cases} \text{rc. } h_1 \rightarrow p:\text{img. } G_1 \\ \text{nc. } h_1 \rightarrow p:\text{img. } G_1 \end{cases} \\
H'_1 &= p? \begin{cases} \text{rc. } p!\text{img. } H_1 \\ \text{nc. } p!\text{img. } H_1 \end{cases} \quad P_1 = h_1! \begin{cases} \text{rc. } h_1?\text{img. } P \\ \text{nc. } h_1?\text{img. } P \end{cases} \quad \diamond
\end{aligned}$$

To formalise the properties of subject reduction and session fidelity [21, 22], the standard LTS for global types can be used.

*Definition 2.11 (LTS for Global Types).* The labelled transition system (LTS) for global types is specified by the rules in Figure 4.

Rule [ICOMM] makes sense since, in a global type  $r \rightarrow s : \{\lambda_i.G_i\}_{i \in I}$ , behaviours involving participants  $p$  and  $q$ , ready to interact with each other uniformly in all branches, can do so if neither of them is involved in a previous interaction between  $r$  and  $s$ . In this case, the interaction between  $p$  and  $q$  is independent of the choice of  $r$ , and may be executed before it. For example (omitting final End) we have

$$r \rightarrow s : \{\lambda_1.p \rightarrow q : \lambda, \lambda_2.p \rightarrow q : \lambda\} \xrightarrow{p\lambda q} r \rightarrow s : \{\lambda_1, \lambda_2\}$$

Subject reduction ensures that the transitions of well-typed sessions are mimicked by those of global types.

**THEOREM 2.12 (SUBJECT REDUCTION [4]).** *If*

$$G \vdash \mathbb{M} \text{ and } \mathbb{M} \xrightarrow{p\lambda q} \mathbb{M}'$$

*then*  $G \xrightarrow{p\lambda q} G' \text{ and } G' \vdash \mathbb{M}'$ .

This theorem requires boundedness of global types: for example, if  $P = q!\{\lambda_1.0, \lambda_2.P\}$  and  $Q = p?\{\lambda_1.0, \lambda_2.Q\}$ , then

$$p[P] \parallel q[Q] \parallel r[s!\lambda.0] \parallel s[r?\lambda.0] \xrightarrow{r\lambda s} p[P] \parallel q[Q]$$

but the type  $G = p \rightarrow q : \{\lambda_1.r \rightarrow s:\lambda.\text{End}, \lambda_2.G\}$  does not have the same reduction. Clearly this session can be typed by the bounded type  $r \rightarrow s:\lambda.G'$  where  $G' = p \rightarrow q : \{\lambda_1.\text{End}, \lambda_2.G'\}$ .

Session fidelity ensures that the communications in a session typed by a global type proceed as prescribed by the global type.

**THEOREM 2.13 (SESSION FIDELITY [4]).** *If*  $G \vdash \mathbb{M}$  *and*  $G \xrightarrow{p\lambda q} G'$ , *then*  $\mathbb{M} \xrightarrow{p\lambda q} \mathbb{M}'$  *and*  $G' \vdash \mathbb{M}'$ .

Note that, if Rule [COMM] had allowed more branches in the global type than in the output process as the subtyping of [17] does, then Theorem 2.13 would have failed. An example is

$$p \rightarrow q : \{\lambda.\text{End}, \lambda'.\text{End}\} \vdash p[q!\lambda.0] \parallel q[p?\{\lambda.0, \lambda'.0\}]$$

since  $p \rightarrow q : \{\lambda.\text{End}, \lambda'.\text{End}\} \xrightarrow{p\lambda' q} \text{End}$ , but there is no transition labelled  $p\lambda' q$  from  $p[q!\lambda.0] \parallel q[p?\{\lambda.0, \lambda'.0\}]$ .

Typability does ensure lock-freedom.

**THEOREM 2.14 (LOCK-FREEDOM [4]).** *If*  $\mathbb{M}$  *is typable, then*  $\mathbb{M}$  *is lock-free.*

We notice that global types, as presented in this section, ensure properties of closed multiparty sessions, where all participant behaviours are fully described,

### 3 MULTICOMPOSITION AND MULTICOMPATIBILITY

As discussed in the Introduction, in the present paper we extend the PaI approach to the PaI multicomposition of closed sessions. In order to exemplify the notions we introduce and their related formal definitions, we shall recur to the following example where we consider four sessions we wish to compose.

*Example 3.1 (Four multiparty sessions).* Let  $\mathbb{M}_1$  be as in Example 2.3, and let us consider also the following multiparty sessions  $\mathbb{M}_2$ ,  $\mathbb{M}_3$  and  $\mathbb{M}_4$ .

*Session*  $\mathbb{M}_2 = h_2[H_2] \parallel q[Q]$ . Process  $H_2$  controls an image display. Images are provided by process  $Q$  according to some parameters with sender  $H_2$  depending on the reaction acquired by a sensor driven by  $q$  and distinguishing the kind of customers on the basis of their cards. Process  $Q$  is also able to receive a `RESET` message even if  $H_2$  cannot ever send it.  $H_2$  and  $Q$  can hence be implemented as follows.

$$H_2 = q?\text{react}.q!\text{pars}.q! \begin{cases} \text{rc. } q?\text{img. } H_2 \\ \text{nc. } q?\text{img. } H_2 \end{cases}$$

$$Q = h_2!\text{react}.h_2?\text{pars}.h_2? \begin{cases} \text{rc. } h_2!\text{img. } Q \\ \text{nc. } h_2!\text{img. } Q \\ \text{reset. } Q \end{cases}$$

*Session*  $\mathbb{M}_3 = h_3[H_3] \parallel r[R] \parallel r'[R']$ . Process  $R$  controls a sensor detecting the entrance of people from a door. Once someone enters, a message `START` is sent by  $R$  to process  $H_3$  which turns on a light. The reaction of who enters, detected by a sensor driven by  $H_3$  is sent back to  $R$ , which, according to the reaction, communicates to  $R'$  the greeting to be broadcasted from the speakers.

$$H_3 = r?\text{start}.r!\text{react}.H_3$$

$$R = h_3!\text{start}.h_3?\text{react}.r'!\text{greet}.R \quad R' = r?\text{greet}.R'$$

*Session*  $\mathbb{M}_4 = h_4[H_4] \parallel s[S]$ . Some sensors managed by process  $H_4$  do acquire the first reactions of people getting into a hall with several Christmas lights. This reactions enable process  $S$  to send to  $H_4$  a set of parameters allowing to adjust the lights of the hall.

$$H_4 = s!\text{react}.s?\text{pars}.H_4 \quad S = h_4?\text{react}.h_4!\text{pars}.S \quad \diamond$$

We shall prove that lock-freedom, ensured by typing on single sessions, is preserved by composition. The sessions of the above example are lock-free.

*Example 3.2.* The multiparty session  $\mathbb{M}_1$  of Example 2.3 can be typed by the global type  $G_1$  of Example 2.10, and the multiparty

sessions  $\mathbb{M}_2, \mathbb{M}_3$  and  $\mathbb{M}_4$  can be typed by the following global types.

$$G_2 = q \rightarrow h_2:\text{REACT}. h_2 \rightarrow q:\text{PARS}. h_2 \rightarrow q: \begin{cases} \text{RC}. q \rightarrow h_2:\text{IMG}. G_2 \\ \text{NC}. q \rightarrow h_2:\text{IMG}. G_2 \end{cases}$$

$$G_3 = r \rightarrow h_3:\text{START}. h_3 \rightarrow r:\text{REACT}. r \rightarrow r':\text{GREET}. G_3$$

$$G_4 = h_4 \rightarrow s:\text{REACT}. s \rightarrow h_4:\text{PARS}. G_4 \quad \diamond$$

The PaI multicomposition consists in replacing one participant per session identified as “interface” by a “gateway” (sort of forwarder). Any participant in a session, say  $h$ , can be considered as an interface. In particular, we can look at the behaviour (a process in our formalism) of  $h$  as what the session would expect from a number of outer sessions (through their respective interfaces). By looking at  $h$  as an interface then, whenever  $h$  receives (resp. sends) a message  $\lambda$ , this has to be interpreted as a message to be sent to (resp. to be received from) some other interface among the available ones.

*Example 3.3 (Interfaces).* For the sessions of Example 3.1 we shall consider the participants  $h_1, h_2, h_3$  and  $h_4$  as interfaces for, respectively, the sessions  $\mathbb{M}_1, \mathbb{M}_1, \mathbb{M}_1$  and  $\mathbb{M}_4$ .  $\diamond$

By having several sessions, the gateways are not uniquely determined. In order to produce gateways out of interfaces we need to decide how the interfaces do interact. We hence call “interfacing policy” a description of a possible way interfaces could communicate with each other. To formalise such a notion we first associate to each process a set of processes doing dual communications with participants taken from a fixed set. We call “interfacing set” this set of processes.

*Definition 3.4 (Interfacing Set).* The *interfacing set* of a process  $H$  w.r.t. a finite set  $\mathcal{P}$  of participants, notation  $\text{IS}(H, \mathcal{P})$ , is the minimal set of processes such that:

- $0 \in \text{IS}(0, \mathcal{P})$ ;
- if  $K_i \in \text{IS}(H_i, \mathcal{P})$  for all  $i \in I$  and  $p \in \mathcal{P}$ , then  $p! \{\lambda_i.K_i\}_{i \in I} \in \text{IS}(q? \{\lambda_i.H_i\}_{i \in I}, \mathcal{P})$
- if  $K_i \in \text{IS}(H_i, \mathcal{P})$  for all  $i \in I$  and  $p \in \mathcal{P}$ , then  $p? \{\lambda_i.K_i\}_{i \in I} \in \text{IS}(q! \{\lambda_i.H_i\}_{i \in I}, \mathcal{P})$

*Example 3.5 (Interfacing set of  $H_1$ ).* Let  $H_1$  be as in Example 2.3 and  $\mathcal{P} = \{h_2, h_3, h_4\}$ , then the interfacing set of  $H_1$  contains all and only the processes

$$K = k_1? \text{START}. k_2! \text{REACT}. k_3! \begin{cases} \text{RC}. k_4? \text{IMG}. K \\ \text{NC}. k_5? \text{IMG}. K \end{cases}$$

where  $k_1, k_2, k_3, k_4, k_5 \in \{h_2, h_3, h_4\}$ .  $\diamond$

Interfacing sets are finite, since they contain processes which only differ for the names of participants and these names belong to a finite set.

An interfacing policy is then obtained by choosing, for each interface, an element of its interfacing set having as participants the other interfaces. Of course one cannot expect an arbitrary interfacing policy to lead to a sound composition. Let us consider, for example, the sessions of Example 3.1 and an interfacing policy where we choose the following element of  $\text{IS}(H_1, \{h_2, h_3, h_4\})$ :

$$K = h_2? \text{START}. h_3! \text{REACT}. h_4! \begin{cases} \text{RC}. h_2? \text{IMG}. K \\ \text{NC}. h_2? \text{IMG}. K \end{cases}$$

This would lead to a composition where the gateway we substitute for  $h_1$  would first expect from  $h_2$  the message  $\text{START}$  to be forwarded to  $p$ . Such a composition would immediately get stuck, since no message  $\text{START}$  is ever handled by  $H_2$  and hence by the gateway we would substitute for it. Sound compositions will actually be the one induced by typable interfacing policies, which we dub as “valid”.

*Definition 3.6 (Interfacing Policy).* An *interfacing policy*  $\mathbb{K}$  for a multiparty session  $\Pi_{i \in I} h_i[H_i]$  is a multiparty session  $\Pi_{i \in I} h_i[K_i]$  such that  $K_i \in \text{IS}(H_i, \mathcal{P} \setminus \{h_i\})$  for all  $i \in I$ , where  $\mathcal{P} = \{h_i \mid i \in I\}$ . An interfacing policy is *valid* if  $\mathbb{K}$  is typable.

*Example 3.7 (Interfacing policies).* Let us consider the four sessions of Example 3.1. Then an interfacing policy for the multiparty session  $\Pi_{i=1}^4 h_i[H_i]$  is the multiparty session  $\Pi_{i=1}^4 h_i[K_i]$  where

$$K_1 = h_3? \text{START}. h_4! \text{REACT}. h_2! \begin{cases} \text{RC}. h_2? \text{IMG}. K_1 \\ \text{NC}. h_2? \text{IMG}. K_1 \end{cases}$$

$$K_2 = h_3! \text{REACT}. h_4? \text{PARS}. h_1? \begin{cases} \text{RC}. h_1! \text{IMG}. K_2 \\ \text{NC}. h_1! \text{IMG}. K_2 \end{cases}$$

$$K_3 = h_1! \text{START}. h_2? \text{REACT}. K_3 \quad K_4 = h_1? \text{REACT}. h_2! \text{PARS}. K_4$$

This policy is valid, since the multiparty session  $\Pi_{i=1}^4 h_i[K_i]$  can be typed by the following global type

$$G = h_3 \rightarrow h_1:\text{START}. h_2 \rightarrow h_3:\text{REACT}. h_1 \rightarrow h_4:\text{REACT}. \hat{G}$$

where

$$\hat{G} = h_4 \rightarrow h_2:\text{PARS}. h_1 \rightarrow h_2: \begin{cases} \text{RC}. h_2 \rightarrow h_1:\text{IMG}. G \\ \text{NC}. h_2 \rightarrow h_1:\text{IMG}. G \end{cases}$$

Note that, according to the above interfacing policy, the greeting depends on the reactions sent by the sensor driven by  $q$ . It is not difficult to check that there exists another valid interfacing policy for  $\Pi_{i=1}^4 h_i[H_i]$ , namely the one according to which the greeting depends on the reactions sent by the sensor driven by  $p$ . I.e. also  $\Pi_{i=1}^4 h_i[K'_i]$  is an interfacing policy for the multiparty session  $\Pi_{i=1}^4 h_i[H_i]$  where

$$K'_1 = h_3? \text{START}. h_3! \text{REACT}. h_2! \begin{cases} \text{RC}. h_2? \text{IMG}. K'_1 \\ \text{NC}. h_2? \text{IMG}. K'_1 \end{cases}$$

$$K'_2 = h_4! \text{REACT}. h_4? \text{PARS}. h_1? \begin{cases} \text{RC}. h_1! \text{IMG}. K'_2 \\ \text{NC}. h_1! \text{IMG}. K'_2 \end{cases}$$

$$K'_3 = h_1! \text{START}. h_1? \text{REACT}. K'_3 \quad K'_4 = h_2? \text{REACT}. h_2! \text{PARS}. K'_4$$

This policy is valid, since the multiparty session  $\Pi_{i=1}^4 h_i[K'_i]$  can be typed by the following global type

$$G' = h_3 \rightarrow h_1:\text{START}. h_1 \rightarrow h_3:\text{REACT}. h_2 \rightarrow h_4:\text{REACT}. \hat{G}'$$

where

$$\hat{G}' = h_4 \rightarrow h_2:\text{PARS}. h_1 \rightarrow h_2: \begin{cases} \text{RC}. h_2 \rightarrow h_1:\text{IMG}. G' \\ \text{NC}. h_2 \rightarrow h_1:\text{IMG}. G' \end{cases} \quad \diamond$$

For a given multiparty session the number of interfacing policies is finite, since interfacing sets are finite.

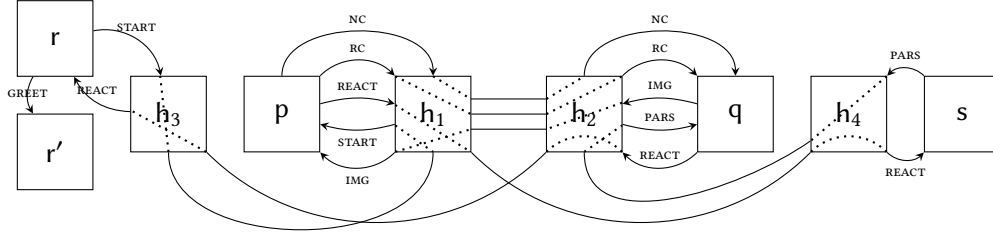


Figure 5: Representation of the composed session in Example 3.13.

Our PaI multicomposition requires that the sessions to be composed be multicompatible. We say that multiparty sessions are multicompatible if they are typable and their participants are disjoint and we identify an interface for each of them and a corresponding valid interfacing policy.

**Definition 3.8 (Multicompatibility).** The multiparty sessions  $\{\mathbb{M}_i\}_{i \in I}$  are *multicompatible with respect to*  $\{h_i\}_{i \in I}$  and  $\mathbb{K}$  if they are typable and  $\text{prt}(\mathbb{M}_j) \cap \text{prt}(\mathbb{M}_l) = \emptyset$  for all  $j, l \in I, j \neq l$ , and  $h_i[H_i] \in \mathbb{M}_i$  for all  $i \in I$  and  $\mathbb{K}$  is a valid interfacing policy for  $\Pi_{i \in I} h_i[H_i]$ .

**Example 3.9 (Multicompatible sessions).** Example 3.7 shows that  $\Pi_{i=1}^4 h_i[K_i]$  is an interfacing policy for  $\Pi_{i=1}^4 h_i[H_i]$ . Such a policy is valid, therefore the multiparty sessions  $\{\mathbb{M}_i\}_{i \in \{1,2,3,4\}}$  are multicompatible with respect to  $\{h_i\}_{i \in \{1,2,3,4\}}$  and  $\Pi_{i=1}^4 h_i[K_i]$ .  $\diamond$

We have almost all the required notions to define the multicomposition of multicompatible multiparty sessions. The only missing piece is that of building the gateways, using the operation of process composition. Two processes can be composed only if they offer exactly matching outputs and inputs: in the composition the inputs always precede the outputs.

**Definition 3.10 (Process Composition).** The *partial composition of two processes*  $P$  and  $Q$ , notation  $P \circ Q$ , is the commutative operator defined by

$$0 \circ 0 = 0 \quad p! \{ \lambda_i. P_i \}_{i \in I} \circ q? \{ \lambda_i. Q_i \}_{i \in I} = q? \{ \lambda_i. p! \lambda_i. P_i \circ Q_i \}_{i \in I}$$

**Example 3.11 (Composition of  $H_1$  and  $K_1$ ).** Let  $H_1$  be as in Example 2.3 and  $K_1$  be as in Example 3.7, then

$$H_1 \circ K_1 = h_3? \text{START}. p! \text{START}. p? \text{REACT}. h_4! \text{REACT}. HK_1$$

$$HK_1 = p? \begin{cases} \text{RC}. h_2! \text{RC}. h_2? \text{IMG}. p! \text{IMG}. H_1 \circ K_1 \\ \text{NC}. h_2! \text{NC}. h_2? \text{IMG}. p! \text{IMG}. H_1 \circ K_1 \end{cases} \quad \diamond$$

**Definition 3.12 (PaI Multicomposition).** Let the multiparty sessions  $\{\mathbb{M}_i\}_{i \in I}$  be multicompatible with respect to  $\{h_i\}_{i \in I}$  and  $\mathbb{K}$ . We define the *PaI multicomposition* of  $\{\mathbb{M}_i\}_{i \in I}$  with respect to  $\{h_i\}_{i \in I}$  and  $\mathbb{K}$  by

$$\Pi_{i \in I} h_i[H_i \circ K_i] \parallel \mathbb{M}'_i$$

where  $\mathbb{M}_i \equiv h_i[H_i] \parallel \mathbb{M}'_i$  for all  $i \in I$  and  $\mathbb{K} = \Pi_{i \in I} h_i[K_i]$ .

In the above definition,  $h_i[H_i \circ K_i]$  are the gateways connecting the multicompatible sessions.

**Example 3.13 (A multicomposition of sessions).** Let  $\mathbb{M}_1, \mathbb{M}_2, \mathbb{M}_3$  and  $\mathbb{M}_4$  be as in Example 3.1. In Example 3.9 it is shown that these multiparty sessions are multicompatible. A PaI multicomposition of the multiparty sessions  $\{\mathbb{M}_i\}_{i \in \{1,2,3,4\}}$  is

$$\Pi_{i=1}^4 h_i[H_i \circ K_i] \parallel p[P] \parallel q[Q] \parallel r[R] \parallel r'[R'] \parallel s[S]$$

where  $H_1, P$  are defined in Example 2.3,  $H_2, H_3, H_4, Q, R, R', S$  are defined in Example 3.1,  $K_1, K_2, K_3, K_4$  are defined in Example 3.7,  $H_1 \circ K_1$  is defined in Example 3.11 and

$$H_2 \circ K_2 = q? \text{REACT}. h_3! \text{REACT}. h_4? \text{PARS}. q! \text{PARS}. HK_2$$

$$HK_2 = h_1? \begin{cases} \text{RC}. q! \text{RC}. q? \text{IMG}. h_1! \text{IMG}. H_2 \circ K_2 \\ \text{NC}. q! \text{NC}. q? \text{IMG}. h_1! \text{IMG}. H_2 \circ K_2 \end{cases}$$

$$H_3 \circ K_3 = r? \text{START}. h_1! \text{START}. h_2? \text{REACT}. r! \text{REACT}. H_3 \circ K_3$$

$$H_4 \circ K_4 = h_1? \text{REACT}. s! \text{REACT}. s? \text{PARS}. h_2! \text{PARS}. H_4 \circ K_4$$

As done for the example in the Introduction, by abstracting from the way communications are performed, from branching and from the logical order of the exchanged messages, the above composition can be graphically described as in Figure 5. It is worth noticing that getting rid of the gateways, so that, e.g.  $r$  sends message `START` directly to  $p$ , would disrupt the *conservativity* of our composition method. Participants other than the interfaces would in fact be affected by the composition, since a number of input/output actions should be modified.  $\diamond$

**Remark 3.14.** The definitions of interfacing set and interfacing policy have a set of participants as parameter. By using a set of messages as extra parameter, we could turn the gateways from “forwarders” to “message-rename-and-forward” processes, so adding extra flexibility to our composition method. The extension is easy and we did not make it explicit for the sake of readability.  $\diamond$

The following lemma relates the global type of an interfacing policy with the global types of two among the multicompatible sessions. It says how the outermost communication in the global type of the interface policy can be related to the outermost communications in the global types of the involved interfaces. This result is crucial for the correctness of our PaI multicomposition. By  $\text{hd}(G)$  we denote the two participants to the outermost communication in  $G$ , i.e. if  $G = p \rightarrow q : \{\lambda_j. G_j\}_{j \in J}$ , then we define  $\text{hd}(G) = \{p, q\}$ .

$$\mathcal{G}(\langle G_i \rangle_{i \in I}, G) = \begin{cases} p \rightarrow h_{i_l} : \{\lambda_j. h_{i_l} \rightarrow h_{i_{l'}} : \lambda_j. h_{i_{l'}} \rightarrow q : \lambda_j. G_j^* \}_{j \in J'} & \text{if } G_{i_m} \text{ is locked for all } i_m \text{ s.t. } m < \min\{l, l'\} \\ \quad \text{where } G_j^* = \mathcal{G}(\langle G_i \rangle_{i \in I \setminus \{i_l, i_{l'}\}} \bullet G_j' \bullet G_j'', \hat{G}_j) & \text{and } G = h_{i_l} \rightarrow h_{i_{l'}} : \{\lambda_j. \hat{G}_j\}_{j \in J} \\ & \text{and } G_{i_l} = p \rightarrow h_{i_l} : \{\lambda_j. G_j'\}_{j \in J'} \\ & \text{and } G_{i_{l'}} = h_{i_{l'}} \rightarrow q : \{\lambda_j. G_j''\}_{j \in J''} \text{ with } J' \subseteq J \subseteq J'' \\ p \rightarrow q : \{\lambda_j. \mathcal{G}(\langle G_i \rangle_{i \in I \setminus \{i_l\}} \bullet G_j', G) \}_{j \in J} & \text{if } h_{i_l} \notin \{p, q\} \\ & \text{and } G_{i_m} \text{ is locked for all } i_m \text{ s.t. } m < l \\ & \text{and } G_{i_l} = p \rightarrow q : \{\lambda_j. G_j'\}_{j \in J} \\ \text{End} & \text{if } I = \emptyset \text{ and } G = \text{End} \end{cases}$$

Figure 6: The global type for multicomposition.

$$\begin{aligned} G' &= r \rightarrow h_3 : \text{START}. h_3 \rightarrow h_1 : \text{START}. h_1 \rightarrow p : \text{START}. q \rightarrow h_2 : \text{REACT}. h_2 \rightarrow h_3 : \text{REACT}. h_3 \rightarrow r : \text{REACT}. \\ p &\rightarrow h_1 : \text{REACT}. h_1 \rightarrow h_4 : \text{REACT}. h_4 \rightarrow s : \text{REACT}. s \rightarrow h_4 : \text{PARS}. h_4 \rightarrow h_2 : \text{PARS}. h_2 \rightarrow q : \text{PARS}. r \rightarrow r' : \text{GREET}. \\ p &\rightarrow h_1 : \begin{cases} \text{RC}. h_1 \rightarrow h_2 : \text{RC}. h_2 \rightarrow q : \text{RC}. q \rightarrow h_2 : \text{IMG}. h_2 \rightarrow h_1 : \text{IMG}. h_1 \rightarrow p : \text{IMG}. G' \\ \text{NC}. h_1 \rightarrow h_2 : \text{NC}. h_2 \rightarrow q : \text{NC}. q \rightarrow h_2 : \text{IMG}. h_2 \rightarrow h_1 : \text{IMG}. h_1 \rightarrow p : \text{IMG}. G' \end{cases} \end{aligned}$$

Figure 7: A type for multicomposition of Example 3.13.

LEMMA 3.15 (RELATIONS BETWEEN TYPES OF INTERFACING POLICIES AND OF MULTICOMPATIBLE SESSIONS). *Let the multiparty sessions  $\{\mathbb{M}_i\}_{i \in I}$  be multicompatible with respect to  $\{h_i\}_{i \in I}$  and  $\mathbb{K}$ . Moreover, let  $G_i \vdash \mathbb{M}_i$  for  $i \in I$  and  $G \vdash \mathbb{K}$ .*

*If  $G = h_l \rightarrow h_{l'} : \{\lambda_j. \hat{G}_j\}_{j \in J}$  and  $h_l \in \text{hd}(G_l)$  and  $h_{l'} \in \text{hd}(G_{l'})$ , then  $G_l = p \rightarrow h_l : \{\lambda_j. G_j'\}_{j \in J'}$  for some  $p$  with  $J' \subseteq J$  and  $G_{l'} = h_{l'} \rightarrow q : \{\lambda_j. G_j''\}_{j \in J''}$  for some  $q$  with  $J \subseteq J''$ . Finally,  $H_l = p? \{\lambda_j. H_j'\}_{j \in J}$  and  $H_{l'} = q! \{\lambda_j. H_j''\}_{j \in J''}$ .*

PROOF. Let  $\mathbb{K} = \Pi_{i \in I} h_i[K_i]$  and  $G = h_l \rightarrow h_{l'} : \{\lambda_j. \hat{G}_j\}_{j \in J}$ , then Rule [COMM] must be applied to derive  $G \vdash \mathbb{K}$  and this implies  $K_l = h_{l'}! \{\lambda_j. K_j'\}_{j \in J}$  and  $K_{l'} = h_l? \{\lambda_j. K_j''\}_{j \in J''}$  with  $J \subseteq J''$ . By Definition 3.6 we get  $H_l = p? \{\lambda_j. H_j'\}_{j \in J}$  for some  $p$  and  $H_{l'} = q! \{\lambda_j. H_j''\}_{j \in J''}$  for some  $q$ . From  $h_l \in \text{hd}(G_l)$  and  $G_l \vdash \mathbb{M}_l$  we get  $G_l = p \rightarrow h_l : \{\lambda_j. G_j'\}_{j \in J'}$  with  $J' \subseteq J$ . From  $h_{l'} \in \text{hd}(G_{l'})$  and  $G_{l'} \vdash \mathbb{M}_{l'}$  we get  $G_{l'} = h_{l'} \rightarrow q : \{\lambda_j. G_j''\}_{j \in J''}$ .  $\square$

We can now show that PaI multicomposition of multicompatible sessions is safe, since it can be typed. This is done by defining a function  $\mathcal{G}$  with two arguments: a list of global types and a global type. This function, applied to the list of the global types of the sessions to be composed and to the global type of a valid interfacing policy witnessing their multicompatibility, returns a global type for the PaI multicomposition.

By  $\langle G_i \rangle_{i \in I}$  we denote the list  $\langle G_{i_1}, G_{i_2}, \dots, G_{i_n} \rangle$  if  $I = \{i_1, i_2, \dots, i_n\}$ .

The addition of a global type at the end of a list of global types, notation  $\langle G_{i_1}, \dots, G_{i_n} \rangle \bullet G$ , does not add the End type. Formally we define

$$\langle G_{i_1}, \dots, G_{i_n} \rangle \bullet G = \begin{cases} \langle G_{i_1}, \dots, G_{i_n}, G \rangle & \text{if } G \neq \text{End}, \\ \langle G_{i_1}, \dots, G_{i_n} \rangle & \text{otherwise.} \end{cases}$$

A global type  $G_{i_l}$  is *locked* for  $G$  in the list  $\langle G_i \rangle_{i \in I}$  if  $i_l \in I$  and  $h_{i_l} \in \text{hd}(G_{i_l})$  and either  $h_{i_l} \notin \text{hd}(G)$  or  $\text{hd}(G) = \{h_{i_l}, h_{i_{l'}}\}$  and  $h_{i_{l'}} \notin \text{hd}(G_{i_{l'}})$ . In words, a global type is locked if its first communication involves the interface but this communication cannot be done since:

- either the interface is not involved in the first communication of the global type for the interfacing policy;
- or the interface is involved in the first communication of the global type for the interfacing policy, but the communicating interface is not involved in the first communication of the global type for the corresponding session.

The function  $\mathcal{G}$  (defined in Figure 6) returns the “merge” of the global types provided as first argument, inserting also the interactions corresponding to the forwarding of the messages sent to the interfaces, as described by the global type provided as second argument. The construction of such a “merge” proceeds coinductively according to the three clauses of the definition.

The first clause applies when the first unlocked global type in the list has an outermost communication involving an interface, and this interface occurs in the outermost communication of  $G$  together with an interface which occurs in the outermost communication of the corresponding global type in the list. The global type of the composition starts with the communication having as sender a participant which is not an interface (as prescribed by  $G_{i_l}$ ), followed by the forwarding between the two involved interfaces as prescribed by  $G$  and then by the communication of the message from the interface which just received it to a participant which is not an interface (as prescribed by  $G_{i_{l'}}$ ). The protocol continues by applying the function  $\mathcal{G}$  to the global types obtained from  $G_{i_l}$ ,  $G_{i_{l'}}$  and  $G$  by erasing the communications done. The continuations of  $G_{i_l}$  and  $G_{i_{l'}}$  are added at the end of the list (where  $G_{i_l}$  and  $G_{i_{l'}}$  have been erased) which becomes the first arguments in the applications of  $\mathcal{G}$ , while the continuations of  $G$  become the second arguments in the applications of  $\mathcal{G}$ .

$$\begin{array}{c}
\frac{G_j^* \vdash h_{i_l'} [H_j'' \circ K_j''] \parallel q[Q_j] \parallel \dots \quad \forall j \in J'}{h_{i_l'} \rightarrow q : \lambda_j. G_j^* \vdash h_{i_l} [H_j' \circ K_j'] \parallel h_{i_l'} [q! \lambda_j. H_j'' \circ K_j''] \parallel q[h_{i_l'}? \{\lambda_j. Q_j\}_{j \in J''}] \parallel \dots \quad \forall j \in J'} \\
\frac{h_{i_l} \rightarrow h_{i_l'} : \lambda_j. h_{i_l'} \rightarrow q : \lambda_j. G_j^* \vdash p[P_j] \parallel h_{i_l} [h_{i_l'}! \lambda_j. H_j' \circ K_j'] \parallel h_{i_l'} [h_{i_l'}? \{\lambda_j. q! \lambda_j. H_j'' \circ K_j''\}_{j \in J''}] \parallel \dots \quad \forall j \in J'}{p \rightarrow h_{i_l} : \{\lambda_j. h_{i_l} \rightarrow h_{i_l'} : \lambda_j. h_{i_l'} \rightarrow q : \lambda_j. G_j^* \vdash p[P_j]\}_{j \in J'} \parallel h_{i_l} [p? \{\lambda_j. h_{i_l'}! \lambda_j. H_j' \circ K_j'\}_{j \in J}] \parallel \dots}
\end{array}$$

Figure 8: Derivation used in the proof of Theorem 3.17.

The second clause applies when the first unlocked global type in the list has an outermost communication not involving interfaces. The global type of the composition starts with this communication and then continues applying the function  $\mathcal{G}$  to the list obtained by erasing  $G_{i_l}$  and by adding the continuations of  $G_{i_l}$  at the end (as first argument) and to  $G$  (as second argument).

The third clause applies when the list is empty and  $G = \text{End}$ .

*Example 3.16 (A type for multicomposition).* The global type of the PaI multicomposition of Example 3.13 can be obtained by applying  $\mathcal{G}$  as defined in Figure 6 to  $\langle G_1, G_2, G_3, G_4 \rangle$  and  $G$ , where  $G_1$  is defined in Example 2.10,  $G_2, G_3, G_4$  are defined in Example 3.1 and  $G$  is defined in Example 3.7. Figure 7 shows the resulting global type.  $\diamond$

Note that *the function  $\mathcal{G}$  is well defined*, since the equations in clauses 1 and 2 are productive, i.e. they always unfold at least one constructor.

We have now the necessary machinery to show the safety of our session multicomposition.

**THEOREM 3.17 (TYPABILITY OF PAI MULTICOMPOSITION).** *If the multiparty sessions  $\{\mathbb{M}_i\}_{i \in I}$  are multicompatible with respect to  $\{h_i\}_{i \in I}$  and  $\mathbb{K}$ , then the PaI multicomposition of  $\{\mathbb{M}_i\}_{i \in I}$  with respect to  $\{h_i\}_{i \in I}$  and  $\mathbb{K}$  is typable.*

**PROOF.** Let  $\mathbb{M}_i \equiv h_i[H_i] \parallel \mathbb{M}_i'$  and  $G_i \vdash \mathbb{M}_i$  for all  $i \in I$  and  $\mathbb{K} = \Pi_{i \in I} h_i[K_i]$  and  $G \vdash \mathbb{K}$ . We prove

$$\mathcal{G}(\langle G_i \rangle_{i \in I}, G) \vdash \Pi_{i \in I} h_i[H_i \circ K_i] \parallel \mathbb{M}_i'$$

The choice of the order in making the list  $\langle G_{i_1}, G_{i_2}, \dots, G_{i_n} \rangle$  is arbitrary, but taking into account that  $\text{prt}(\mathbb{M}_j) \cap \text{prt}(\mathbb{M}_l) = \emptyset$  implies  $\text{prt}(G_j) \cap \text{prt}(G_l) = \emptyset$  for all  $j, l \in I, j \neq l$ , typability of sessions by the obtained global type is insensible to this order.

*The obtained type  $\mathcal{G}(\langle G_i \rangle_{i \in I}, G)$  is bounded*, provided that  $\mathcal{G}$  is total. The proof is by coinduction on  $G_i$  for  $i \in I$  and on  $G$  and by cases on the three clauses. Note that all  $G_i$  for  $i \in I$  and  $G$  are bounded since they type multiparty sessions.

**Clause 1.** By coinduction  $\mathcal{G}(\langle G_i \rangle_{i \in I \setminus \{i_l, i_{l'}\}} \bullet G_j' \bullet G_j'', \hat{G}_j)$  is bounded for all  $j \in J'$ ;

**Clause 2.** By coinduction  $\mathcal{G}(\langle G_i \rangle_{i \in I \setminus \{i_l\}} \bullet G_j', G)$  is bounded for all  $j \in J$ ;

**Clause 3.** Trivial.

We now show by coinduction that, in case  $\mathcal{G}$  is total, we can derive

$$\mathcal{G}(\langle G_i \rangle_{i \in I}, G) \vdash \Pi_{i \in I} h_i[H_i \circ K_i] \parallel \mathbb{M}_i'$$

We proceed by cases according to which among the three clauses defining  $\mathcal{G}$  is applied.

**Clause 1.** Lemma 3.15 implies  $H_{i_l} = p? \{\lambda_j. H_j'\}_{j \in J}$ .

From  $G_{i_l} \vdash \mathbb{M}_{i_l}$  we get

$$\mathbb{M}_{i_l} \equiv p[h_{i_l}! \{\lambda_j. P_j\}_{j \in J'}] \parallel h_{i_l} [p? \{\lambda_j. H_j'\}_{j \in J}] \parallel \mathbb{M}_{i_l}''$$

with  $J' \subseteq J$  and

$$G_j' \vdash p[P_j] \parallel h_{i_l} [H_j'] \parallel \mathbb{M}_{i_l}'' \text{ for all } j \in J' \quad (1)$$

Lemma 3.15 implies  $H_{i_{l'}} = q! \{\lambda_j. H_j''\}_{j \in J''}$  with  $J \subseteq J''$ . From  $G_{i_{l'}} \vdash \mathbb{M}_{i_{l'}}$  we get

$$\mathbb{M}_{i_{l'}} \equiv h_{i_{l'}} [q! \{\lambda_j. H_j''\}_{j \in J''}] \parallel q[h_{i_{l'}}? \{\lambda_j. Q_j\}_{j \in J''}] \parallel \mathbb{M}_{i_{l'}}''$$

with  $J'' \subseteq J'''$  and

$$G_j'' \vdash h_{i_{l'}} [H_j''] \parallel q[Q_j] \parallel \mathbb{M}_{i_{l'}}'' \text{ for all } j \in J'' \quad (2)$$

From  $G \vdash \Pi_{i \in I} h_i[K_i]$  and  $K_{i_l} \in \text{IS}(H_{i_l}, \{h_i\}_{i \in I \setminus \{i_l\}})$  and  $K_{i_{l'}} \in \text{IS}(H_{i_{l'}}, \{h_i\}_{i \in I \setminus \{i_{l'}\}})$  we get

$$K_{i_l} = h_{i_{l'}}! \{\lambda_j. K_j'\}_{j \in J} \quad K_{i_{l'}} = h_{i_l}? \{\lambda_j. K_j''\}_{j \in J''}$$

and

$$\hat{G}_j \vdash h_{i_l} [K_j'] \parallel h_{i_{l'}} [K_j''] \parallel \Pi_{i \in I \setminus \{i_l, i_{l'}\}} h_i[K_i] \text{ for all } j \in J \quad (3)$$

The definition of interfacing policy (Definition 3.6) implies that, for all  $j \in J'$ ,

$$\begin{aligned}
&h_{i_l} [K_j'] \parallel h_{i_{l'}} [K_j''] \parallel \Pi_{i \in I \setminus \{i_l, i_{l'}\}} h_i[K_i] \\
&\text{is an interfacing policy for} \\
&h_{i_l} [H_j'] \parallel h_{i_{l'}} [H_j''] \parallel \Pi_{i \in I \setminus \{i_l, i_{l'}\}} h_i[H_i]
\end{aligned} \quad (4)$$

Then the PaI multicomposition is

$$h_{i_l} [HK_{i_l}] \parallel h_{i_{l'}} [HK_{i_{l'}}] \parallel \Pi_{i \in I \setminus \{i_l, i_{l'}\}} h_i[H_i \circ K_i] \parallel \Pi_{i \in I} \mathbb{M}_i'$$

where

$$\begin{aligned}
HK_{i_l} &= p? \{\lambda_j. h_{i_{l'}}! \lambda_j. H_j' \circ K_j'\}_{j \in J} \\
HK_{i_{l'}} &= h_{i_l}? \{\lambda_j. q! \lambda_j. H_j'' \circ K_j''\}_{j \in J''}
\end{aligned}$$

By coinduction the typings (1), (2), (3) and the statement (4) imply that, for all  $j \in J'$ ,

$$G_j^* \vdash p[P_j] \parallel h_{i_l} [H_j' \circ K_j'] \parallel h_{i_{l'}} [H_j'' \circ K_j''] \parallel q[Q_j] \parallel \mathbb{M}'$$

where  $\mathbb{M}' \equiv \mathbb{M}_{i_l}'' \parallel \mathbb{M}_{i_{l'}}'' \parallel \Pi_{i \in I \setminus \{i_l, i_{l'}\}} h_i[H_i \circ K_i] \parallel \Pi_{i \in I \setminus \{i_l, i_{l'}\}} \mathbb{M}_i'$ . We have then the derivation given in Figure 8, where we only show the processes which are modified from the premises to the conclusion.

**Clause 2.** In this case from  $G_{i_l} \vdash \mathbb{M}_{i_l}$  we get

$$\mathbb{M}_{i_l} \equiv p[q! \{\lambda_j. P_j\}_{j \in J}] \parallel q[p? \{\lambda_j. Q_j\}_{j \in J'}] \parallel h_{i_l} [H_{i_l}] \parallel \mathbb{M}_{i_l}''$$

with  $J \subseteq J'$  and

$$G_j' \vdash p[P_j] \parallel q[Q_j] \parallel h_{i_l} [H_{i_l}] \parallel \mathbb{M}_{i_l}'' \text{ for all } j \in J \quad (5)$$

By coinduction, the typing (5) implies for all  $j \in J$

$$\hat{G}_j \vdash p[P_j] \parallel q[Q_j] \parallel \mathbb{M}_{i_l}'' \parallel \Pi_{i \in I} h_i[H_i \circ K_i] \parallel \Pi_{i \in I \setminus \{i_l\}} \mathbb{M}_i'$$

where  $\hat{G}_j = \mathcal{G}(\langle G_i \rangle_{i \in I \setminus \{i_j\}} \bullet G'_j, G)$ . Then we can derive

$$\frac{\hat{G}_j \vdash p[P_j] \parallel q[Q_j] \parallel \dots \quad \forall j \in J}{p \rightarrow q : \lambda_j. \hat{G}_j \vdash p[q!\{\lambda_j.P_j\}_{j \in J}] \parallel q[p?\{\lambda_j.Q_j\}_{j \in J'}] \parallel \dots}$$

where we only show the processes which are modified from the premises to the conclusion.

**Clause 3. Trivial.**

We conclude the proof by showing that *the function  $\mathcal{G}$  is total*. Note that, when clause 1 is applied, the typings (1), (2), (3) and the statement (4) imply that the conditions required by Lemma 3.15 remain valid for the global types  $G'_j, G''_j, \hat{G}_j$  and the corresponding multiparty sessions for all  $j \in J'$ . When clause 2 is applied the typing (5) implies that the conditions required by Lemma 3.15 remain valid for the global types  $G'_j$  and the corresponding multiparty sessions for all  $j \in J$ .

Let  $G_{i_m}$  be locked for  $G$  in the list  $\langle G_i \rangle_{i \in I}$  for all  $m < l$  and  $G_{i_l}$  be unlocked. If  $h_{i_l} \in \text{hd}(G_{i_l})$  and  $\text{hd}(G) = \{i_l, i_{l'}\}$  for some  $i_{l'} \in I$  and  $h_{i_{l'}} \in \text{hd}(G_{i_{l'}})$ , then Lemma 3.15 ensures that  $G_{i_l}, G_{i_{l'}}$  and  $G$  have the shapes required in clause 1, possibly exchanging inputs and outputs. If  $h_{i_l} \notin \text{hd}(G_{i_l})$ , then clause 2 applies. Lastly  $I = \emptyset$  implies  $G = \text{End}$ , so clause 3 applies.  $\square$

## 4 TYPE INFERENCE

The effectiveness of PaI multicomposition relies on the following facts:

- (1) the possible choices of participants to be replaced by gateways are finite;
- (2) there is a finite number of interfacing policies for a given session;
- (3) global types for sessions can be inferred, if any.

Facts (1) and (2) are clear from the previous sections. In this section we describe an algorithm to infer global types for sessions, by adapting to synchronous communication the algorithm of [16] in order to handle matching of input and output processes.

Since global types are regular terms, we represent them as finite systems of regular syntactic equations [1, 15]. We prove soundness and completeness of the algorithm with respect to the typing system: when applied to a session  $\mathbb{M}$ , it finds all and only those global types that can be derived for  $\mathbb{M}$ , if any. Note that, since a session may have more than one global type, to be complete, the algorithm needs to be non-deterministic.

The algorithm follows the structure of coSLD resolution of coinductive logic programming [2, 33–35], namely the extension of SLD resolution capable to deal with regular infinite terms and coinductive predicates. The key idea, borrowed from coinductive logic programming, is to keep track of already encountered variables to detect cycles and avoid non-termination.

A *global-type pattern* is a finite term generated by the following grammar.

$$\mathbb{G} ::= \text{End} \mid p \rightarrow q : \{\lambda_i. \mathbb{G}_i\}_{i \in I} \mid X$$

where  $X$  is a variable taken from a countably infinite set. We denote by  $\text{vars}(\mathbb{G})$  the set of variables occurring in  $\mathbb{G}$ . A *substitution*  $\theta$  is a finite partial map from variables to global types. We denote by

$\mathbb{G}\theta$  the application of  $\theta$  to  $\mathbb{G}$ . Note that, if  $\text{vars}(\mathbb{G}) \subseteq \text{dom}(\theta)$ , then  $\mathbb{G}\theta$  is a global type. An *equation* has shape  $X = \mathbb{G}$  and a (*regular*) *system of equations*  $\mathcal{E}$  is a finite set of equations such that  $X = \mathbb{G}_1$  and  $X = \mathbb{G}_2 \in \mathcal{E}$  imply  $\mathbb{G}_1 = \mathbb{G}_2$ . We denote by  $\text{vars}(\mathcal{E})$  the set  $\{X \mid X = \mathbb{G} \in \mathcal{E}\}$ . A *solution* of a system  $\mathcal{E}$  is a substitution  $\theta$  such that  $\text{vars}(\mathcal{E}) \subseteq \text{dom}(\theta)$  and, for all  $X = \mathbb{G} \in \mathcal{E}$ ,  $\theta(X) = \mathbb{G}\theta$  holds. We denote by  $\text{sol}(\mathcal{E})$  the set of all solutions of  $\mathcal{E}$ . Note that  $\mathcal{E}_1 \subseteq \mathcal{E}_2$  implies  $\text{sol}(\mathcal{E}_2) \subseteq \text{sol}(\mathcal{E}_1)$ .

The algorithm takes in input a *goal* (a pair  $(X, \mathbb{M})$ ) and either fails or returns a set of equations  $\mathcal{E}$  such that the solution for the variable  $X$  in  $\mathcal{E}$  is a global type for the session  $\mathbb{M}$ . Rules defining the inference algorithm are reported in Figure 9. Inference judgements are of the shape  $S \vdash (X, \mathbb{M}) \Rightarrow \mathcal{E}$ , where  $S$  is a set of goals; variables in  $S$  are pairwise distinct and different from  $X$ .

For a terminated session the algorithm returns one equation  $X = \text{End}$  (Rule [A-END]). For other sessions (Rule [A-COMM]) the algorithm selects one of the matching pairs:  $P = q!\{\lambda_i.P_i\}_{i \in I}$  and  $Q = p?\{\lambda_j.Q_j\}_{j \in J}$ , with  $I \subseteq J$ . The algorithm continues analysing all matching branches  $P_i$  and  $Q_i$ . After having evaluated subsessions, the algorithm collects all the resulting equations plus another one for the current variable. The freshness condition on variables  $Y_i$  ensures that the resulting set  $\mathcal{E}$  is a regular system of equations. The side condition on participants ensures that the resulting global type associated with  $X$  satisfies the conditions on participants required by Rule [COMM] in Definition 2.8. The set  $\text{prt}(S; \mathcal{E}; \mathbb{G})$  is defined as the set of participants of a global type, but with the following additional clause to handle variables:

$$\text{prt}(S; \mathcal{E}; X) = \begin{cases} \text{prt}(S; \mathcal{E}; \mathbb{G}) & \text{if } X = \mathbb{G} \in \mathcal{E} \\ \text{prt}(\mathbb{M}) & \text{if } X \notin \text{dom}(\mathcal{E}) \text{ and } (X, \mathbb{M}) \in S \\ \emptyset & \text{otherwise} \end{cases}$$

Finally, Rule [A-CYCLE] detects cycles: if the session in the current goal appears also in  $S$ , the algorithm can stop and return just one equation that unifies two variables.

*Example 4.1 (Inference).* Figure 10 shows the application of the rules of Figure 9 to the session of Example 2.3, where

$$\begin{aligned} S_1 &= (X, h_1[H_1] \parallel p[P]) \\ S_2 &= S_1, (Y_1, h_1[p?\text{REACT}. H'_1] \parallel p[h_1!\text{REACT}. P_1]) \\ S_3 &= S_2, (Y_2, h_1[H'_1] \parallel p[P_1]) \\ S_4 &= S_3, (Y_3, h_1[p!\text{IMG}. H_1] \parallel p[h_1?\text{IMG}. P]) \\ S_5 &= S_4, (Y_4, h_1[p!\text{IMG}. H_1] \parallel p[h_1?\text{IMG}. P]) \\ \mathcal{E}_6 &= \{Y_6 = X\} \\ \mathcal{E}_5 &= \{Y_5 = X\} \\ \mathcal{E}_4 &= \{Y_4 = p \rightarrow h_1!\text{IMG}. Y_6\} \cup \mathcal{E}_6 \\ \mathcal{E}_3 &= \{Y_3 = p \rightarrow h_1!\text{IMG}. Y_5\} \cup \mathcal{E}_5 \\ \mathcal{E}_2 &= \{Y_2 = p \rightarrow h_1:\{\text{rc}. Y_3, \text{nc}. Y_4\}\} \cup \mathcal{E}_3 \cup \mathcal{E}_4 \\ \mathcal{E}_1 &= \{Y_1 = p \rightarrow h_1!\text{REACT}. Y_2\} \cup \mathcal{E}_2 \\ \mathcal{E} &= \{X = h_1 \rightarrow p:\text{START}. Y_1\} \cup \mathcal{E}_1 \end{aligned}$$

The sets of goals and equations above are listed according to the order in which they are produced in a possible execution of the algorithm implicitly described by the rules of Figure 9. It is easy to verify that a solution is the global type given in Example 2.10. It is then useful to compare Figure 10 with Figure 3.  $\diamond$

$$\begin{array}{c}
\text{[A-CYCLE]} \frac{}{\mathcal{S}, (Y, \mathbb{M}) \vdash (X, \mathbb{M}) \Rightarrow \{X = Y\}} \quad \text{[A-END]} \frac{}{\mathcal{S} \vdash (X, p[0]) \Rightarrow \{X = \text{End}\}} \\
\text{[A-COMM]} \frac{\mathcal{S}' \vdash (Y_i, p[P_i] \parallel q[Q_i] \parallel \mathbb{M}) \Rightarrow \mathcal{E}_i \quad \forall i \in I}{\mathcal{S} \vdash (X, p[q!\{\lambda_i.P_i\}_{i \in I}] \parallel q[p?\{\lambda_j.Q_j\}_{j \in J}] \parallel \mathbb{M}) \Rightarrow \mathcal{E}} \quad \begin{array}{l} \mathcal{S}' = \mathcal{S}, (X, p[P] \parallel q[Q] \parallel \mathbb{M}) \quad I \subseteq J \\ Y_i \text{ fresh } \forall i \in I \quad \mathcal{E} = \{X = p \rightarrow q : \{\lambda_i.Y_i\}_{i \in I}\} \cup \bigcup_{i \in I} \mathcal{E}_i \\ \text{prt}(\mathcal{S}'; \mathcal{E}_i; Y_i) \setminus \{p, q\} = \text{prt}(\mathbb{M}) \quad \forall i \in I \end{array}
\end{array}$$

Figure 9: Rules of the inference algorithm.

$$\begin{array}{c}
\frac{}{\mathcal{S}_4 \vdash (Y_5, h_1[H_1] \parallel p[P]) \Rightarrow \mathcal{E}_5} \text{[A-CYCLE]} \quad \frac{}{\mathcal{S}_5 \vdash (Y_6, h_1[H_1] \parallel p[P]) \Rightarrow \mathcal{E}_6} \text{[A-CYCLE]} \\
\frac{\mathcal{S}_3 \vdash (Y_3, h_1[p!_{\text{IMG}}.H_1] \parallel p[h_1?_{\text{IMG}}.P]) \Rightarrow \mathcal{E}_3}{\mathcal{S}_2 \vdash (Y_2, h_1[H'_1] \parallel p[P_1]) \Rightarrow \mathcal{E}_2} \text{[A-COMM]} \quad \frac{\mathcal{S}_3 \vdash (Y_4, h_1[p!_{\text{IMG}}.H_1] \parallel p[h_1?_{\text{IMG}}.P]) \Rightarrow \mathcal{E}_4}{\mathcal{S}_1 \vdash (Y_1, h_1[p?_{\text{REACT}}.H'_1] \parallel p[h_1!_{\text{REACT}}.P_1]) \Rightarrow \mathcal{E}_1} \text{[A-COMM]} \\
\frac{\mathcal{S}_1 \vdash (Y_1, h_1[p?_{\text{REACT}}.H'_1] \parallel p[h_1!_{\text{REACT}}.P_1]) \Rightarrow \mathcal{E}_1}{\vdash (X, h_1[H_1] \parallel p[P]) \Rightarrow \mathcal{E}} \text{[A-COMM]}
\end{array}$$

Figure 10: Type inference for the session of Example 2.3.

Some definitions are handy. We denote by  $\theta + \sigma$  the union of two substitutions such that  $\theta(X) = \sigma(X)$ , for all  $X \in \text{dom}(\theta) \cap \text{dom}(\sigma)$ . We denote by  $\text{vars}(\mathcal{E})$  the set  $\bigcup \{\text{vars}(\mathbb{G}) \cup \{X\} \mid X = \mathbb{G} \in \mathcal{E}\}$ . We define  $\theta \leq \sigma$  if  $\text{dom}(\theta) \subseteq \text{dom}(\sigma)$  and  $\theta(X) = \sigma(X)$ , for all  $X \in \text{dom}(\theta)$ . Let  $\mathcal{E}$  be a system of equations and  $\mathcal{S}$  a set of goals. A solution  $\theta \in \text{sol}(\mathcal{E})$  *agrees* with  $\mathcal{S}$  if  $(X, \mathbb{M}) \in \mathcal{S}$  implies  $\text{prt}(\theta(X)) = \text{prt}(\mathbb{M})$  for all  $X \in \text{vars}(\mathcal{E})$ . We denote by  $\text{sol}_{\mathcal{S}}(\mathcal{E})$  the set of all solutions of  $\mathcal{E}$  agreeing with  $\mathcal{S}$ . We say that a system of equations  $\mathcal{E}$  is *guarded* if  $X = Y$  and  $Y = \mathbb{G}$  in  $\mathcal{E}$  imply that  $\mathbb{G}$  is not a variable. Finally,  $\mathcal{E}$  is  *$\mathcal{S}$ -closed* if it is guarded and  $\text{dom}(\mathcal{E}) \cap \text{vars}(\mathcal{S}) = \emptyset$  and  $\text{vars}(\mathcal{E}) \setminus \text{dom}(\mathcal{E}) \subseteq \text{vars}(\mathcal{S})$ .

Toward proving properties of the inference algorithm, we check a couple of auxiliary lemmas.

As usual  $\mathcal{S} \vdash (X, \mathbb{M}) \Rightarrow \mathcal{E}$  means that this judgment belongs to a derivation in the system of Figure 9 having a judgment with an empty sets of goals as conclusion (namely it represents the result of a recursive call during the execution of our algorithm).

LEMMA 4.2. *If  $\mathcal{S} \vdash (X, \mathbb{M}) \Rightarrow \mathcal{E}$ , then  $\mathcal{E}$  is  $\mathcal{S}$ -closed.*

PROOF. By induction on the derivation of  $\mathcal{S} \vdash (X, \mathbb{M}) \Rightarrow \mathcal{E}$ .  $\square$

LEMMA 4.3. *If  $\mathcal{E}$  is an  $\mathcal{S}$ -closed system of equations and  $\text{vars}(\mathbb{G}) \subseteq \text{vars}(\mathcal{E})$ , then  $\text{prt}(\mathcal{S}; \mathcal{E}; \mathbb{G}) = \text{prt}(\mathbb{G}\theta)$  for all  $\theta \in \text{sol}_{\mathcal{S}}(\mathcal{E})$ .*

PROOF. To prove the inclusion  $\text{prt}(\mathbb{G}\theta) \subseteq \text{prt}(\mathcal{S}; \mathcal{E}; \mathbb{G})$ , let  $p \in \text{prt}(\mathbb{G}\theta)$ . We show  $p \in \text{prt}(\mathcal{S}; \mathcal{E}; \mathbb{G})$  by induction on the least distance  $d$  of a communication with player  $p$  from the root of  $\mathbb{G}\theta$ . First of all, it is easy to see that there is  $\mathbb{G}'$  such that  $\text{prt}(\mathcal{S}; \mathcal{E}; \mathbb{G}) = \text{prt}(\mathcal{S}; \mathcal{E}; \mathbb{G}')$  and  $\mathbb{G}\theta = \mathbb{G}'\theta$  and either  $\mathbb{G}' = r \rightarrow s : \{\lambda_i.\mathbb{G}_i\}_{i \in I}$  or  $\mathbb{G}' = X$  and  $X \notin \text{dom}(\mathcal{E})$ . Indeed, we have  $\mathbb{G} \neq \text{End}$  since  $\text{prt}(\text{End}\theta) = \text{prt}(\text{End}) = \emptyset$ . First we show that  $\mathbb{G} = X \in \text{dom}(\mathcal{E})$  is impossible. In this case  $X = \mathbb{G}_1 \in \mathcal{E}$  and we have  $\mathbb{G}\theta = \mathbb{G}_1\theta$  and  $\text{prt}(\mathcal{S}; \mathcal{E}; \mathbb{G}) = \text{prt}(\mathcal{S}; \mathcal{E}; \mathbb{G}_1)$ , since  $\theta$  is a solution of  $\mathcal{E}$ . Hence, again  $\mathbb{G}_1 \neq \text{End}$  and if  $\mathbb{G}_1 = Y \in \text{dom}(\mathcal{E})$ , namely,  $Y = \mathbb{G}_2 \in \mathcal{E}$ , we have  $\mathbb{G}_1\theta = \mathbb{G}_2\theta$  and  $\text{prt}(\mathcal{S}; \mathcal{E}; \mathbb{G}_1) = \text{prt}(\mathcal{S}; \mathcal{E}; \mathbb{G}_2)$  and, since  $\mathcal{E}$  is  $\mathcal{S}$ -closed and so guarded, we have that  $\mathbb{G}_2$  is not a variable.

**Case  $d = 0$ .** If  $\mathbb{G}' = X \notin \text{dom}(\mathcal{E})$ , then  $(X, \mathbb{M}) \in \mathcal{S}$  and  $\text{prt}(\mathcal{S}; \mathcal{E}; \mathbb{G}') = \text{prt}(\mathbb{M})$ . Since  $\theta$  agrees with  $\mathcal{S}$ , we have  $\text{prt}(\mathbb{G}'\theta) =$

$\text{prt}(\theta(X)) = \text{prt}(\mathbb{M})$ , hence  $p \in \text{prt}(\mathcal{S}; \mathcal{E}; \mathbb{G}')$ .

If  $\mathbb{G}' = r \rightarrow s : \{\lambda_i.\mathbb{G}_i\}_{i \in I}$ , then  $\mathbb{G}'\theta = r \rightarrow s : \{\lambda_i.\mathbb{G}_i\theta\}_{i \in I}$  and  $p \in \text{prt}(\mathbb{G}') = \text{prt}(\mathbb{G}'\theta)$ . By definition we have  $\text{prt}(\mathcal{S}; \mathcal{E}; \mathbb{G}') = \text{prt}(\mathcal{S}') \cup \bigcup_{i \in I} \text{prt}(\mathcal{S}, \mathcal{E}, \mathbb{G}_i)$ , hence  $p \in \text{prt}(\mathcal{S}; \mathcal{E}; \mathbb{G}')$ .

**Case  $d > 0$ .** If  $\mathbb{G}' = X \notin \text{dom}(\mathcal{E})$ , the proof is as above. If  $\mathbb{G}' = r \rightarrow s : \{\lambda_i.\mathbb{G}_i\}_{i \in I}$ , then  $p \notin \text{hd}(\mathbb{G}'\theta)$ , hence  $p \notin \{r, s\}$ . We have  $\mathbb{G}'\theta = r \rightarrow s : \{\lambda_i.\mathbb{G}_i\theta\}_{i \in I}$  and there is  $l \in I$  such that  $p \in \text{prt}(\mathbb{G}_l\theta)$  and the distance decreases. Then, by induction hypothesis, we get  $p \in \text{prt}(\mathcal{S}; \mathcal{E}; \mathbb{G}_l) \subseteq \text{prt}(\mathcal{S}; \mathcal{E}; \mathbb{G}')$ , as needed.

To prove the other inclusion,  $\text{prt}(\mathcal{S}; \mathcal{E}; \mathbb{G}) \subseteq \text{prt}(\mathbb{G}\theta)$ , we just have to check that the sets  $\text{prt}(\mathbb{G}\theta)$  respect the equations defining  $\text{prt}(\mathcal{S}; \mathcal{E}; \mathbb{G})$ . All cases are trivial except for  $\mathbb{G} = X$ . If  $X \in \text{dom}(\mathcal{E})$ , that is,  $X = \mathbb{G}' \in \mathcal{E}$ , then  $\mathbb{G}\theta = \theta(X) = \mathbb{G}'\theta$ , hence  $\text{prt}(\mathbb{G}\theta) = \text{prt}(\mathbb{G}'\theta)$ , as needed. Otherwise,  $X \in \text{vars}(\mathcal{S})$ , that is,  $(X, \mathbb{M}) \in \mathcal{S}$ , hence  $\text{prt}(\mathcal{S}; \mathcal{E}; \mathbb{G}) = \text{prt}(\mathbb{M})$ . Since  $\theta$  agrees with  $\mathcal{S}$ , we have  $\text{prt}(\mathbb{G}\theta) = \text{prt}(\theta(X)) = \text{prt}(\mathbb{M})$ , as needed.  $\square$

To show soundness and completeness of our inference algorithm, it is handy to formulate an inductive version of our typing rules, see Figure 11, where  $\mathcal{N}$  ranges over sets of pairs  $(\mathbb{M}, \mathbb{G})$ . We can give an inductive formulation since all infinite derivations using the typing rules of Definition 2.8 are regular, i.e. the number of different subtrees of a derivation for a judgement  $\mathbb{G} \vdash \mathbb{M}$  is finite. In fact, it is bounded by the product of the number of different subterms of  $\mathbb{G}$  and the number of different subnetworks of  $\mathbb{M}$ , which are both finite as  $\mathbb{G}$  and (processes in)  $\mathbb{M}$  are regular. Applying the standard transformation according to [31, Section 21.9] from a coinductive to an inductive formulation we get the typing rules shown in Figure 11.

*Example 4.4 (Inductive formulation).* The inductive formulation of the derivation in Figure 3 is shown in Figure 12, where

$$\begin{aligned}
\mathcal{N}' &= (h_1[H_1] \parallel p[P], G_1), \\
&\quad (h_1[p?_{\text{REACT}}.H'_1] \parallel p[h_1!_{\text{REACT}}.P_1], p \rightarrow h_1!_{\text{REACT}}.G'_1), \\
&\quad (h_1[H'_1] \parallel p[P_1], G'_1) \\
\mathcal{N} &= \mathcal{N}', (h_1[p!_{\text{IMG}}.H_1] \parallel p[h_1?_{\text{IMG}}.P], h_1 \rightarrow p!_{\text{IMG}}.G_1) \quad \diamond
\end{aligned}$$

$$\begin{array}{c}
\text{[I-CYCLE]} \frac{}{\mathcal{N}, (\mathbb{M}, G) \vdash_i \mathbb{M} : G} \quad \text{[I-END]} \frac{}{\mathcal{N} \vdash_i p[0] : \text{End}} \\
\text{[I-COMM]} \frac{\mathcal{N}, (\mathbb{M}, G) \vdash_i p[P_i] \parallel q[Q_i] \parallel \mathbb{M}' : G_i \quad \text{prt}(G_i) \setminus \{p, q\} = \text{prt}(\mathbb{M}') \quad \forall i \in I}{\mathcal{N} \vdash_i \mathbb{M} : G} \quad G = p \rightarrow q : \{\lambda_i. G_i\}_{i \in I} \\
\mathbb{M} = p[q!\{\lambda_i. P_i\}_{i \in I}] \parallel q[p?\{\lambda_j. Q_j\}_{j \in J}] \parallel \mathbb{M}'
\end{array}$$

Figure 11: Inductive typing rules for sessions.

$$\begin{array}{c}
\frac{\mathcal{N} \vdash_i h_1[H_1] \parallel p[P] : G_1}{\mathcal{N}' \vdash_i h_1[p!_{\text{IMG}}. H_1] \parallel p[h_1?_{\text{IMG}}. P] : h_1 \rightarrow p!_{\text{IMG}}. G_1} \quad \frac{\mathcal{N} \vdash_i h_1[H_1] \parallel p[P] : G_1}{\mathcal{N}' \vdash_i h_1[p!_{\text{IMG}}. H_1] \parallel p[h_1?_{\text{IMG}}. P] : h_1 \rightarrow p!_{\text{IMG}}. G_1} \\
\frac{(h_1[H_1] \parallel p[P], G_1), (h_1[p?_{\text{REACT}}. H'_1] \parallel p[h_1!_{\text{REACT}}. P_1], p \rightarrow h_1!_{\text{REACT}}. G'_1) \vdash_i h_1[H'_1] \parallel p[P_1] : G'_1}{(h_1[H_1] \parallel p[P], G_1) \vdash_i h_1[p?_{\text{REACT}}. H'_1] \parallel p[h_1!_{\text{REACT}}. P_1] : p \rightarrow h_1!_{\text{REACT}}. G'_1} \\
\vdash_i h_1[H_1] \parallel p[P] : G_1
\end{array}$$

Figure 12: Inductive derivation for Example 2.10.

In the following two lemmas we relate inference and inductive derivability.

**LEMMA 4.5.** *If  $\mathcal{S} \vdash (X, \mathbb{M}) \Rightarrow \mathcal{E}$  and  $\theta(X)$  is bounded, then  $\mathcal{S}\theta \vdash_i \mathbb{M} : \theta(X)$*

for all  $\theta \in \text{sol}_{\mathcal{S}}(\mathcal{E})$  such that  $\text{vars}(\mathcal{S}) \subseteq \text{dom}(\theta)$ .

**PROOF.** By induction on the derivation of  $\mathcal{S} \vdash (X, \mathbb{M}) \Rightarrow \mathcal{E}$ .

**Rule [A-END].** We have  $\mathcal{E} = \{X = \text{End}\}$ , hence  $\theta(X) = \text{End}$  and the thesis follows by Rule [I-END].

**Rule [A-CYCLE].** We have  $\mathcal{E} = \{X = Y\}$  and  $\mathcal{S} = \mathcal{S}', (Y, \mathbb{M})$ . Then,  $\theta(X) = \theta(Y)$  and the thesis follows by Rule [I-CYCLE].

**Rule [A-COMM].** We have

$$\mathbb{M} \equiv p[q!\{\lambda_i. P_i\}_{i \in I}] \parallel q[p?\{\lambda_j. Q_j\}_{j \in J}] \parallel \mathbb{M}'$$

with  $I \subseteq J$  and  $\mathcal{S}, (X, \mathbb{M}) \vdash (Y_i, \mathbb{M}_i) \Rightarrow \mathcal{E}_i$  with  $Y_i$  fresh and  $\mathbb{M}_i \equiv p[P_i] \parallel q[Q_i] \parallel \mathbb{M}'$  and  $\text{prt}(\mathcal{S}, (X, \mathbb{M}); \mathcal{E}_i; Y_i) \setminus \{p, q\} = \text{prt}(\mathbb{M}')$  for all  $i \in I$  and  $\mathcal{E} = \{X = p \rightarrow q : \{\lambda_i. Y_i\}_{i \in I}\} \cup \bigcup_{i \in I} \mathcal{E}_i$ . Since  $\mathcal{E}_i \subseteq \mathcal{E}$ , we have  $\theta \in \text{sol}(\mathcal{E}_i)$ . Being  $\theta \in \text{sol}_{\mathcal{S}}(\mathcal{E})$ , Lemma 4.3 implies  $\text{prt}(\mathcal{S}; \mathcal{E}; X) = \text{prt}(\mathbb{M})$ . So we get that  $\theta$  agrees with  $\mathcal{S}, (X, \mathbb{M})$ . Then, by induction, we have  $\mathcal{S}\theta, (\mathbb{M}, \theta(X)) \vdash_i \mathbb{M}_i : \theta(Y_i)$  for all  $i \in I$ . The thesis follows by Rule [I-COMM], since

$$\theta(X) = p \rightarrow q : \{\lambda_i. \theta(Y_i)\}_{i \in I}$$

and  $\text{prt}(\mathcal{S}, (X, \mathbb{M}); \mathcal{E}_i; Y_i) \setminus \{p, q\} = \text{prt}(\mathbb{M}')$  imply  $\text{prt}(\theta(Y_i)) \setminus \{p, q\} = \text{prt}(\mathbb{M}')$

for all  $i \in I$  by Lemma 4.3.  $\square$

**LEMMA 4.6.** *If  $\mathcal{N} \vdash_i \mathbb{M} : G$  and  $\text{prt}(G') = \text{prt}(\mathbb{M}')$  for all  $(G', \mathbb{M}') \in \mathcal{N}$ , then, for all  $\mathcal{S}, X$  and  $\sigma$  such that  $X \notin \text{vars}(\mathcal{S})$ ,  $\text{dom}(\sigma) = \text{vars}(\mathcal{S})$  and  $\mathcal{S}\sigma = \mathcal{N}$ , there are  $\mathcal{E}$  and  $\theta$  such that  $\mathcal{S} \vdash (X, \mathbb{M}) \Rightarrow \mathcal{E}$  and  $\theta \in \text{sol}_{\mathcal{S}}(\mathcal{E})$  and  $\text{dom}(\theta) = \text{vars}(\mathcal{E}) \cup \text{vars}(\mathcal{S})$  and  $\sigma \leq \theta$  and  $\theta(X) = G$ .*

**PROOF.** By induction on the derivation of  $\mathcal{N} \vdash_i \mathbb{M} : G$ .

**Rule [I-END].** The thesis is immediate by Rule [A-END] taking  $\theta = \sigma + \{X \mapsto \text{End}\}$ .

**Rule [I-CYCLE].** We have  $\mathcal{N} = \mathcal{N}', (\mathbb{M}, G)$ , then  $\mathcal{S} = \mathcal{S}', (Y, \mathbb{M})$  and  $\sigma(Y) = G$ . By Rule [A-CYCLE], we get  $\mathcal{S} \vdash (X, \mathbb{M}) \Rightarrow \{X = Y\}$ ,

hence  $\theta = \sigma + \{X \mapsto G\}$  is a solution of  $\{X = Y\}$ , which agrees with  $\mathcal{S}$  being  $\text{prt}(G) = \text{prt}(\mathbb{M})$ , as needed.

**Rule [I-COMM].** In this case we have

$$\mathbb{M} \equiv p[q!\{\lambda_i. P_i\}_{i \in I}] \parallel q[p?\{\lambda_j. Q_j\}_{j \in J}] \parallel \mathbb{M}'$$

with  $I \subseteq J$  and  $G = p \rightarrow q : \{\lambda_i. G_i\}_{i \in I}$  and  $\mathcal{N}, (\mathbb{M}, G) \vdash_i \mathbb{M}_i : G_i$  with  $\mathbb{M}_i \equiv p[P_i] \parallel q[Q_i] \parallel \mathbb{M}'$  and  $\text{prt}(G_i) \setminus \{p, q\} = \text{prt}(\mathbb{M}')$ , for all  $i \in I$ . This last condition implies  $\text{prt}(G) = \text{prt}(\mathbb{M})$ . Set  $\sigma' = \sigma + \{X \mapsto G\}$  and  $\mathcal{S}' = \mathcal{S}, (X, \mathbb{M})$ , then, by induction hypothesis, we get that there are  $\mathcal{E}_i$  and  $\theta_i$  such that  $\mathcal{S}' \vdash (Y_i, \mathbb{M}_i) \Rightarrow \mathcal{E}_i$  and  $\theta_i \in \text{sol}_{\mathcal{S}'}(\mathcal{E}_i)$  and  $\text{dom}(\theta_i) = \text{vars}(\mathcal{E}_i) \cup \text{vars}(\mathcal{S}')$  and  $\sigma' \leq \theta_i$  and  $\theta_i(Y_i) = G_i$ , for all  $i \in I$ . We can assume that  $j \neq l$  implies  $Y_j \neq Y_l$  and  $\text{dom}(\mathcal{E}_j) \cap \text{dom}(\mathcal{E}_l) = \emptyset$  for all  $j, l \in I$ , because the algorithm always introduces fresh variables. This implies  $\text{dom}(\theta_j) \cap \text{dom}(\theta_l) = \text{vars}(\mathcal{S}')$  for all  $j \neq l$ , and so  $\theta = \sum_{i \in I} \theta_i$  is well defined. Moreover, we have  $\theta \in \text{sol}_{\mathcal{S}'}(\mathcal{E}_i)$  and  $\sigma \leq \theta$  and  $\theta(X) = G$ , as  $\sigma \leq \sigma'$  and  $\sigma' \leq \theta_i \leq \theta$  for all  $i \in I$ . From  $\text{prt}(G_i) \setminus \{p, q\} = \text{prt}(\mathbb{M}')$  we get  $\text{prt}(\mathcal{S}'; \mathcal{E}_i; Y_i) \setminus \{p, q\} = \text{prt}(\mathbb{M}')$  for all  $i \in I$  by Lemma 4.3. By Rule [A-COMM] we get  $\mathcal{S} \vdash (X, \mathbb{M}) \Rightarrow \mathcal{E}$  with

$$\mathcal{E} = \{X = p \rightarrow q : \{\lambda_i. Y_i\}_{i \in I}\} \cup \bigcup_{i \in I} \mathcal{E}_i$$

and  $\theta \in \text{sol}_{\mathcal{S}}(\mathcal{E})$ , since

$$\begin{aligned}
\theta(X) &= p \rightarrow q : \{\lambda_i. G_i\}_{i \in I} = p \rightarrow q : \{\lambda_i. \theta_i(Y_i)\}_{i \in I} \\
&= (p \rightarrow q : \{\lambda_i. Y_i\}_{i \in I})\theta
\end{aligned}$$

and  $\sigma \leq \theta$ .  $\square$

Soundness and completeness state that the inference algorithm applied to a session finds all and only the global types which, if bounded, can be assigned to the session.

**THEOREM 4.7 (SOUNDNESS AND COMPLETENESS OF INFERENCE).**

- (1) If  $\vdash (X, \mathbb{M}) \Rightarrow \mathcal{E}$ , then  $\theta(X) \vdash \mathbb{M}$  for all  $\theta \in \text{sol}(\mathcal{E})$  such that  $\theta(X)$  is bounded.
- (2) If  $G \vdash \mathbb{M}$ , then there are  $\mathcal{E}$  and  $\theta$  such that  $\vdash (X, \mathbb{M}) \Rightarrow \mathcal{E}$  and  $\theta \in \text{sol}(\mathcal{E})$  and  $\theta(X) = G$ .

**PROOF.** (1). By Lemma 4.5  $\vdash (X, \mathbb{M}) \Rightarrow \mathcal{E}$  implies  $\vdash_i \mathbb{M} : \theta(X)$  for all  $\theta \in \text{sol}(\mathcal{E})$ . This is enough, since  $\vdash_i \mathbb{M} : \theta(X)$  gives  $\theta(X) \vdash \mathbb{M}$ .

(2). From  $G \vdash \mathbb{M}$  we get  $\vdash_1 \mathbb{M} : G$ . By Lemma 4.6 this implies that there are  $\mathcal{E}$  and  $\theta$  such that  $\vdash (X, \mathbb{M}) \Rightarrow \mathcal{E}$  and  $\theta \in \text{sol}(\mathcal{E})$  and  $\theta(X) = G$ .  $\square$

*Remark 4.8 (Termination).* To avoid non-termination, the key idea, borrowed from coinductive logic programming, is to keep track of already encountered goals to detect cycles.

As it happens for (co)SLD-resolution in logic programming, the termination of our inference algorithm depends on the choice of a resolution strategy. Indeed, we have many sources of non-determinism: we have to pick two participants of the session with matching processes and expand them using Rule [A-COMM], or try to close a cycle using the Rule [A-CYCLE]. A standard way to obtain a sound and complete resolution strategy is to build a tree where all such choices are performed in parallel and then visit the tree using a breadth-first strategy. The tree is potentially infinite in depth, but it is finitely branching, since at each point we have only finitely many different choices, hence this strategy necessarily finds all solutions. In case no rule can be applied, the algorithm fails.  $\diamond$

## 5 CONCLUSION AND RELATED WORK

In the present paper we have addressed the problem of *multiple* protocol composition for MPSTs [14, 18, 21, 22, 32, 37] using the calculus and the type system defined in [4]. We extended the PaI approach devised in [3] and exploited in [5] for binary composition of MPSTs. By binary composition, however, only tree-like structures can be obtained, leaving out many compositional possibilities. In [4] the PaI approach for MPSTs was adapted to multiple composition in a client-server setting, where many server-sessions can be “connected” to just one client-session. In the present paper, instead, all the sessions to be composed are peers and their gateways can freely interact. This was achieved by introducing the notion of *multicompatibility*, which boils down to identifying a *typable* “interfacing policy” (a session describing how gateways interact). The gateways connecting the various sessions are hence defined in terms of such interfacing policy. We proved that lock-freedom (ensured by typability) is preserved by composition. It is worth remarking that, as shown by a simple working example, one could have many typable interfacing policies to choose among.

A different approach to composition for MPST is taken in [36], where sessions with missing participants can be typed and composed when types are compatible. A limit of that work is that only finite processes are considered.

The formalism of MPSTs used for our investigation can be dubbed as *bottom-up*: no projection is used and sessions are checked against global types by means of a type assignment system. Multicomposition in a *top-down* MPST setting has instead been recently addressed in [19]. In a *top-down* MPST, communication protocols are explicitly described as global types and, subsequently, by projecting them, local types are obtained for implementation. The present paper and [19] address then multicomposition from two orthogonal perspectives. In a sense, however, they both exploit the general idea of ensuring *safe* multicomposition by means of a *safe* interfacing policy. As a matter of fact, the “traditional” syntax of global types is extended in [19] in order to explicitly describe an interfacing policy inside the global types themselves. Its projectability enables hence to apply a composition operation at the global type

level. Unlike our approach, the interfacing policy that “drives” the composition is *rigid* in the sense that it is univocally determined by the global types to be composed. The main advantage of our approach over [19] is hence the possibility of choosing among different interfacing policies for the same set of sessions. On the other hand, however, [19] possesses the relevant and expressive feature of enabling more than a single interface in a session.

In [23] the author devises a type assignment system in logical form for sessions, where just one type is present, processes are unnamed and communications are performed through an (implicit) single channel. Deadlock-freedom is ensured by typability only in case the session enjoys a race-freedom condition. In such a context the composition of two sessions with single interfaces corresponds to a particular form of Cut rule, where compatibility corresponds to duality. Thanks to the presence of a single communication channel and to the absence of process names, sessions can be composed by simply removing the interfaces. It is not possible of course to get any explicit global information about the behaviour of sessions because just one type is present. The setting of the present paper, by using global types, process names and multiple point-to-point channels is however definitely more expressive and realistic. Our notion of compatibility cannot reduce to duality and the use of interfacing policies enables to finely control the operation of composition.

In [11], forwarders are introduced in a Linear Logic interpretation of a MPST formalism. Such forwarders are in a proofs-as-processes correspondence with coherence proofs, where coherence is the multiparty counterpart of (binary) duality. Forwarders can be safely composed through cut elimination, so allowing to “compose” two concurrent sessions. The precise relationship between the forwarders of [11] and our gateways is worth investigating. Besides, our multiple composition through interfacing policies could have a logical counterpart enabling to compose multiple forwarders.

By suitably combining the notions of multicompatibility of the present paper and the one in [19], one could avoid to extend the syntax of global types as done in [19], retaining at the same time the possibility of having several possible interfacing policies to choose among, as in the present paper. Moreover, the result of the present paper could be extended to the case of more than one interface in the sessions to be composed.

In Remark 3.14 a simple (decidable) extension of the gateways, so that they can perform also some “message renaming”, has been discussed. This idea could actually be pushed further, investigating the possibility of reordering messages, so implicitly introducing a form of *asynchronous subtyping* between gateways. This should be done with care, since in general asynchronous subtyping is undecidable, as shown in [10, 28].

The idea underlying our multicomposition is likely to be applicable in future to other frameworks. The works in [26] and [18] explicitly give algorithms for the synthesis of global types from communicating finite-state machines, while [27] proposes a similar method to build graphical choreographies, expressed as global graphs. [32] develops instead a framework where global types are not necessary, relying on model and type checking techniques for verifying safety properties of collections of local types. Our notion of interfacing policy could be adapted to such frameworks. Then we could investigate whether the synthesis of global types/global graphs or the verification of properties via type checking does

“lift” from the components to the composed session, proviso a type synthesis or checking is provided for the chosen interfacing policy.

MPSTs are characterised by the implicit or explicit presence of tools for checking/verifying session properties (type assignment, projectability, type checking, etc.). The application of safe compositional methods can however be investigated independently from such tools. This has been done for the formalism of CFSMs in [3, 6, 7] using the PaI approach for binary composition. Suitable adaptations of the notions of interfacing policy and multicompatability could be hopefully devised in such setting. As future work we also plan to consider composition of MPSTs with asynchronous communications, taking advantage from the more liberal syntax of global types introduced in [12].

## ACKNOWLEDGMENTS

We wish to gratefully thank the anonymous reviewers for their thoughtful and helpful comments. This research was partially funded by EPSRC EP/T006544/2, EP/K011715/1, EP/K034413/1, EP/L00058X/1, EP/N027833/2, EP/N028201/1, EP/T014709/2, EP/V000462/1, EP/X015955/1, NCSS/EPSRC VeTSS and EU Horizon TaRDIS, 101093006. The first author was partially supported by the Project “National Center for HPC, Big Data e Quantum Computing”, Programma M4C2 – dalla ricerca all’impresa – Investimento 1.3: Creazione di “Partenariati estesi alle università, ai centri di ricerca, alle aziende per il finanziamento di progetti di ricerca di base” – Next Generation EU; by the Piano Triennale Ricerca Pia.Ce.Ri UniCT; and by project ATRASIoT.

## REFERENCES

- [1] Jiri Adámek, Stefan Milius, and Jiri Velebil. 2006. Iterative algebras at work. *Mathematical Structures in Computer Science* 16, 6 (2006), 1085–1131. <https://doi.org/10.1017/S0960129506005706>
- [2] Davide Ancona and Agostino Dovier. 2015. A theoretical perspective of coinductive logic programming. *Fundamenta Informaticae* 140, 3-4 (2015), 221–246. <https://doi.org/10.3233/FI-2015-1252>
- [3] Franco Barbanera, Ugo de'Liguoro, and Rolf Hennicker. 2019. Connecting open systems of communicating finite state machines. *Journal of Logical and Algebraic Methods in Programming* 109 (2019), 100476. <https://doi.org/10.1016/j.jlamp.2019.07.004>
- [4] Franco Barbanera, Mariangiola Dezani-Ciancaglini, and Ugo de'Liguoro. 2022. Open compliance in multiparty sessions. In *FACS (LNCS, Vol. 13712)*, S. Lizeth Tapia Tarifa and José Proença (Eds.). Springer, Berlin, 222–243. [https://doi.org/10.1007/978-3-031-20872-0\\_13](https://doi.org/10.1007/978-3-031-20872-0_13) extended version at <http://www.di.unito.it/~dezani/papers/bd23b.pdf>.
- [5] Franco Barbanera, Mariangiola Dezani-Ciancaglini, Ivan Lanese, and Emilio Tuosto. 2021. Composition and decomposition of multiparty sessions. *Journal of Logic and Algebraic Methods in Programming* 119 (2021), 100620. <https://doi.org/10.1016/j.jlamp.2020.100620>
- [6] Franco Barbanera, Ivan Lanese, and Emilio Tuosto. 2020. Composing communicating systems, synchronously. In *ISoLA (LNCS, Vol. 12476)*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer, Berlin, 39–59. [https://doi.org/10.1007/978-3-030-61362-4\\_3](https://doi.org/10.1007/978-3-030-61362-4_3)
- [7] Franco Barbanera, Ivan Lanese, and Emilio Tuosto. 2022. Formal choreographic languages. In *COORDINATION (LNCS, Vol. 13271)*, Maurice H. ter Beek and Marjan Sirjani (Eds.). Springer, Berlin, 121–139. [https://doi.org/10.1007/978-3-031-08143-9\\_8](https://doi.org/10.1007/978-3-031-08143-9_8)
- [8] Franco Barbanera, Ivan Lanese, and Emilio Tuosto. 2022. On composing communicating systems. In *ICE (EPTCS, Vol. 365)*, Clément Aubert, Cinzia Di Giusto, Larisa Safina, and Alceste Scalas (Eds.). Open Publishing Association, Waterloo, 53–68. <https://doi.org/10.4204/EPTCS.365.4>
- [9] Daniel Brand and Pitro Zafropulo. 1983. On communicating finite-state machines. *Journal of ACM* 30, 2 (1983), 323–342. <https://doi.org/10.1145/322374.322380>
- [10] Mario Bravetti, Marco Carbone, and Gianluigi Zavattaro. 2017. Undecidability of asynchronous session subtyping. *Information and Computation* 256 (2017), 300–320. <https://doi.org/10.1016/j.ic.2017.07.010>
- [11] Marco Carbone, Sonia Marin, and Carsten Schürmann. 2021. Synchronous forwarders. *CoRR* abs/2102.04731 (2021), 44 pages. [arXiv:2102.04731](https://arxiv.org/abs/2102.04731)
- [12] Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. 2021. Global types and event structure semantics for asynchronous multiparty sessions. <https://doi.org/10.48550/ARXIV.2102.00865>
- [13] David Castro-Perez, Francisco Ferreira, Lorenzo Gheri, and Nobuko Yoshida. 2021. Zoid: a DSL for certified multiparty computation: from mechanised metatheory to certified multiparty processes. In *PLDI*, Stephen N. Freund and Eran Yahav (Eds.). ACM, New York, NY, 237–251. <https://doi.org/10.1145/3453483.3454041>
- [14] Mario Coppo, Mariangiola Dezani-Ciancaglini, Luca Padovani, and Nobuko Yoshida. 2015. A gentle introduction to multiparty asynchronous session types. In *SFM (LNCS, Vol. 9104)*, Marco Bernardo and Einar Broch Johnsen (Eds.). Springer, Berlin, 146–178. [https://doi.org/10.1007/978-3-319-18941-3\\_4](https://doi.org/10.1007/978-3-319-18941-3_4)
- [15] Bruno Courcelle. 1983. Fundamental properties of infinite trees. *Theoretical Computer Science* 25 (1983), 95–169. [https://doi.org/10.1016/0304-3975\(83\)90059-2](https://doi.org/10.1016/0304-3975(83)90059-2)
- [16] Francesco Dagnino, Paola Giannini, and Mariangiola Dezani-Ciancaglini. 2023. Deconfined global types for asynchronous sessions. *Logical Methods in Computer Science* Volume 19, Issue 1 (2023), 1–41. [https://doi.org/10.46298/lmcs-19\(1:3\)2023](https://doi.org/10.46298/lmcs-19(1:3)2023)
- [17] Romain Demangeon and Kohei Honda. 2012. Nested protocols in session types. In *CONCUR (LNCS, Vol. 7454)*, Maciej Koutny and Irek Ulidowski (Eds.). Springer, Berlin, 272–286. [https://doi.org/10.1007/978-3-642-32940-1\\_20](https://doi.org/10.1007/978-3-642-32940-1_20)
- [18] Pierre-Malo Deniérou and Nobuko Yoshida. 2013. Multiparty compatibility in communicating automata: characterisation and synthesis of global session types. In *ICALP*, Fedor V. Fomin, Rūsis Freivalds, Marta Kwiatkowska, and David Peleg (Eds.). Springer, Berlin, 174–186. [https://doi.org/10.1007/978-3-642-39212-2\\_18](https://doi.org/10.1007/978-3-642-39212-2_18)
- [19] Lorenzo Gheri and Nobuko Yoshida. 2023. Hybrid Multiparty Session Types: Compositionality for Protocol Specification through Endpoint Projection. *Proc. ACM Program. Lang.* 7, OOPSLA1, Article 79 (2023), 31 pages. <https://doi.org/10.1145/3586031>
- [20] Kohei Honda, Aybek Mukhamedov, Gary Brown, Tzu-Chun Chen, and Nobuko Yoshida. 2011. Scribbling interactions with a formal foundation. In *ICDCIT (LNCS, Vol. 6536)*, Raja Natarajan and Adegboyega Ojo (Eds.). Springer, Berlin, 55–75. [https://doi.org/10.1007/978-3-642-19056-8\\_4](https://doi.org/10.1007/978-3-642-19056-8_4)
- [21] Kohei Honda, Nobuko Yoshida, and Marco Carbone. 2008. Multiparty asynchronous session types. In *POPL*, George C. Necula and Philip Wadler (Eds.). ACM Press, New York, NY, 273–284. <https://doi.org/10.1145/1328897.1328472>
- [22] Kohei Honda, Nobuko Yoshida, and Marco Carbone. 2016. Multiparty asynchronous session types. *Journal of the ACM* 63, 1 (2016), 9:1–9:67. <https://doi.org/10.1145/2827695>
- [23] Ross Horne. 2020. Session subtyping and multiparty compatibility using circular sequents. In *CONCUR (LIPIcs, Vol. 171)*, Igor Konnov and Laura Kovács (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl, 12:1–12:22. <https://doi.org/10.4230/LIPIcs.CONCUR.2020.12>
- [24] Naoki Kobayashi. 2002. A type system for lock-free processes. *Information and Computation* 177, 2 (2002), 122–159. <https://doi.org/10.1006/inco.2002.3171>
- [25] Dexter Kozen and Alexandra Silva. 2017. Practical Coinduction. *Mathematical Structures in Computer Science* 27, 7 (2017), 1132–1152. <https://doi.org/10.1017/S0960129515000493>
- [26] Julien Lange and Emilio Tuosto. 2012. Synthesising choreographies from local session types. In *CONCUR (LNCS, Vol. 7454)*, Maciej Koutny and Irek Ulidowski (Eds.). Springer, Berlin, 225–239. [https://doi.org/10.1007/978-3-642-32940-1\\_17](https://doi.org/10.1007/978-3-642-32940-1_17)
- [27] Julien Lange, Emilio Tuosto, and Nobuko Yoshida. 2015. From communicating machines to graphical choreographies. In *POPL*, Sriram K. Rajamani and David Walker (Eds.). ACM Press, New York, NY, 221–232. <https://doi.org/10.1145/2676726.2676964>
- [28] Julien Lange and Nobuko Yoshida. 2017. On the undecidability of asynchronous session subtyping. In *FOSSACS (LNCS, Vol. 10203)*, Javier Esparza and Andrzej S. Murawski (Eds.). Springer, Berlin, 441–457. [https://doi.org/10.1007/978-3-662-54458-7\\_26](https://doi.org/10.1007/978-3-662-54458-7_26)
- [29] Rumyana Neykova and Nobuko Yoshida. 2019. Featherweight Scribble. In *Models, Languages, and Tools for Concurrent and Distributed Programming: Essays Dedicated to Rocco De Nicola on the Occasion of His 65th Birthday (LNCS, Vol. 11665)*, Michele Boreale, Flavio Corradini, Michele Loret, and Rosario Pugliese (Eds.). Springer, Berlin, 236–259. [https://doi.org/10.1007/978-3-030-21485-2\\_14](https://doi.org/10.1007/978-3-030-21485-2_14)
- [30] Luca Padovani. 2014. Deadlock and lock freedom in the linear  $\pi$ -calculus. In *CSL-LICS*, Thomas A. Henzinger and Dale Miller (Eds.). ACM Press, New York, NY, 72:1–72:10. [https://doi.org/10.1007/978-3-662-43376-8\\_10](https://doi.org/10.1007/978-3-662-43376-8_10)
- [31] Benjamin C. Pierce. 2002. *Types and Programming Languages*. MIT Press, Cambridge, MA, 1–XXI, 1–623 pages.
- [32] Alceste Scalas and Nobuko Yoshida. 2019. Less is more: multiparty session types revisited. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 30:1–30:29. <https://doi.org/10.1145/3290343>
- [33] Luke Simon. 2006. *Extending logic programming with coinduction*. Ph.D. Dissertation. University of Texas at Dallas.
- [34] Luke Simon, Ajay Bansal, Ajay Mallya, and Gopal Gupta. 2007. Co-Logic programming: extending logic programming with coinduction. In *ICALP (LNCS, Vol. 4596)*, Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki (Eds.). Springer, Berlin, 472–483. [https://doi.org/10.1007/978-3-540-73420-8\\_42](https://doi.org/10.1007/978-3-540-73420-8_42)

- [35] Luke Simon, Ajay Mallya, Ajay Bansal, and Gopal Gupta. 2006. Coinductive logic programming. In *ICLP (LNCS, Vol. 4079)*, Sandro Etalle and Mirosław Truszczyński (Eds.). Springer, Berlin, 330–345. [https://doi.org/10.1007/11799573\\_25](https://doi.org/10.1007/11799573_25)
- [36] Claude Stölze, Marino Miculan, and Pietro Di Gianantonio. 2023. Composable partial multiparty session types for open systems. *Software and Systems Modeling* 22, 2 (2023), 473–494. <https://doi.org/10.1007/s10270-022-01040-x>
- [37] Nobuko Yoshida and Lorenzo Gheri. 2020. A very gentle introduction to multiparty session types. In *ICDCIT (LNCS, Vol. 11969)*, Dang Van Hung and Meenakshi D’Souza (Eds.). Springer, Berlin, 73–93. [https://doi.org/10.1007/978-3-030-36987-3\\_5](https://doi.org/10.1007/978-3-030-36987-3_5)
- [38] Nobuko Yoshida, Raymond Hu, Romyana Neykova, and Nicholas Ng. 2013. The Scribble protocol language. In *TGC (LNCS, Vol. 8358)*, Martín Abadi and Alberto Lluch-Lafuente (Eds.). Springer, Berlin, 22–41. [https://doi.org/10.1007/978-3-319-05119-2\\_3](https://doi.org/10.1007/978-3-319-05119-2_3)
- [39] Nobuko Yoshida, Fangyi Zhou, and Francisco Ferreira. 2021. Communicating finite state machines and an extensible toolchain for multiparty session types. In *FCT (LNCS, Vol. 12867)*, Evripidis Bampis and Aris Pagourtzis (Eds.). Springer, Berlin, 18–35. [https://doi.org/10.1007/978-3-030-86593-1\\_2](https://doi.org/10.1007/978-3-030-86593-1_2)