Composition and Decomposition of Multiparty Sessions

Franco Barbanera, Mariangiola Dezani-Ciancaglini, Ivan Lanese, Emilio Tuosto

Composition and Decomposition of Multiparty Sessions

Franco Barbanera[a,1,2], Mariangiola Dezani-Ciancaglini[b], Ivan Lanese[c,2], Emilio Tuosto[d,3]

[a] *Dipartimento di Mat. e Inf., Università di Catania (Italy), barba@dmi.unict.it*
[b] *Dipartimento di Informatica, Università di Torino (Italy), dezani@di.unito.it*
[c] *Focus Team, University of Bologna/INRIA (Italy), ivan.lanese@gmail.com*
[d] *Gran Sasso Science Institute (Italy) and University of Leicester (UK), emilio.tuosto@gssi.it*

Abstract

Multiparty sessions are systems of concurrent processes, which allow several participants to communicate by sending and receiving messages. Their overall behaviour can be described by means of global types. Typable multiparty sessions enjoy lock-freedom.

We look at multiparty sessions as *open* systems by allowing one to *compose* multiparty sessions by transforming two of their participants into a pair of coupled *gateways*, forwarding messages between the two sessions. Gateways need to be *compatible*. We show that the session resulting from the composition can be typed, and its type can be computed from the global types of the starting sessions. As a consequence, lock-freedom is preserved by composition. Compatibility between global types is *necessary*, since systems obtained by composing sessions with incompatible global types have locks (or they are not sessions). We also define *direct composition*, which allows one to connect two global types without using gateways. Finally, we propose a *decomposition* operator, to split a global type into two, which is the left inverse of direct composition. Direct composition and decomposition on global types prepare the ground for a novel framework allowing for the modular design and implementation of distributed systems.

[1] Partially supported by the project "Piano Triennale Ricerca" DMI-Università di Catania.

[2] Partially supported by INdAM as members of GNCS (Gruppo Nazionale per il Calcolo Scientifico)

[3] Research partly supported by the EU H2020 RISE programme under the Marie Sklodowska-Curie grant agreement No 778233 and by the MIUR project PRIN 2017FTXR7S "IT-MaTTerS" (Methods and Tools for Trustworthy Smart Systems).

1. Introduction

Distributed systems are seldom developed as independent entities. Indeed, in many cases they should be considered as open entities ready for interaction with an environment. Composition may be specified either statically, for a fixed environment, or dynamically, if the environment only becomes known upon deployment. In general, it is fairly natural to expect to connect open systems as if they were composable modules, and in doing that one should rely on "safe" methodologies and techniques, guaranteeing the composition not to "break" any relevant property of the single systems.

In [1] a methodology has been proposed for the composition of systems, consisting in replacing *any* two participants (one per system) - if their behaviours are "compatible" - by a pair of coupled forwarders, dubbed *gateways*, enabling the two systems to exchange messages. In this approach, the behaviour of any participant can be looked at as an *interface*. In this setting, the notion of interface is interpreted not as the description of the interactions "offered" by a system but, dually, as those "required" by a possible environment (usually another system).

A convenient way to tackle the complexity of message-passing applications is by means of *global specifications*. The idea is to devise an abstract representation of the system, the so-called *global view*, that can be used as the blueprint for the realisation of each participant. A neat formalisation of this idea are MultiParty Session Types (MPSTs), introduced in [31, 32]. The global view is rendered in MPSTs as a *global type* meant to capture the interactions among the participants of the protocol. We illustrate, by means of a simple example, how the general idea of composition of systems by gateways can be exploited in the setting of MPTSs.

Let us assume to have System **1** formed by two participants, Alice and Bob, where Alice sends Bob a *req*uest for some advice and receives back either the *adv*ice asked for or an apology (*sorry*) for not being of any help. Let us also assume to have System **2**, with participants Carol and Donald, where Carol sends a message *req* to Donald who surely replies with an advice, since, in case he does not know what to say, he provides one good for any occasion. Figure 1 shows the UML-like diagrams (following the intuitive notation introduced in [31]) of Systems **1** and **2**, where the curly bracket is used to group possible choices. These diagrams yield the global views of our systems.

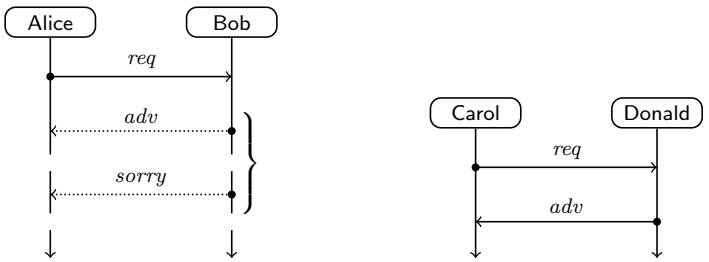


Figure 1: UML-like diagrams of Systems **1** and **2**

The representation in terms of global types of these views is as follows:

$$\mathsf{G}_1 = \mathsf{Alice} \to \mathsf{Bob} : \mathit{req}.\mathsf{Bob} \to \mathsf{Alice} : \{\mathit{adv}.\mathtt{End}, \mathit{sorry}.\mathtt{End}\}$$
$$\mathsf{G}_2 = \mathsf{Carol} \to \mathsf{Donald} : \mathit{req}.\mathsf{Donald} \to \mathsf{Carold} : \mathit{adv}.\mathtt{End}$$

Notice how G_1 and G_2 give a faithful textual representation of the diagrams in Figure 1. An advantage of global types is that they can be equipped with a precise semantics that allows one to verify if a realisation is correct with respect to the specification. For instance, the interaction behaviour of (a realisation of) Bob and of Carol can be specified respectively as:

$$\mathsf{Alice}?\mathit{req}.\mathsf{Alice}!\{\mathit{adv},\ \mathit{sorry}\} \qquad \text{and} \qquad \mathsf{Donald}!\mathit{req}.\mathsf{Donald}?\mathit{adv} \qquad (1)$$

where ?_ means input, !_ means output and between curly brackets we put possible input or output choices (those for input corresponding to *external* choices and those for output to *internal* choices).

Let us now think of System **1** and System **2** as open systems. Then, rather than looking at Bob and Carol as actual participants, we can consider them as *interfaces*. More precisely, the realisations in (1) represent the behaviour of advice-providing and advice-requesting interfaces of systems to which System **2** and System **1**, respectively, are willing to connect. Since any request can be satisfied[4], System **1** and System **2** can be connected by simply replacing participants Bob and Carol, respectively, by the following "forwarding" processes (dubbed *gateways*):

$$\mathsf{Alice}?\mathit{req}.\mathsf{Carol}!\mathit{req}.\mathsf{Carol}?\{\mathit{adv}.\mathsf{Alice}!\mathit{adv},\ \mathit{sorry}.\mathsf{Alice}!\mathit{sorry}\}$$
$$\mathsf{Bob}?\mathit{req}.\mathsf{Donald}!\mathit{req}.\mathsf{Donald}?\mathit{adv}.\mathsf{Bob}!\mathit{adv}$$

[4] As a matter of fact, in [1] "requests" and "offers" are required to exactly match, whereas in the present paper such a restriction can be safely relaxed.

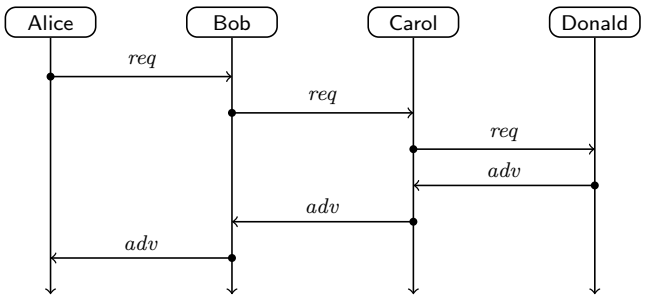


Figure 2: UML-like diagram of Systems **1** and **2** gateway composition

Notice that Bob's option of sending an apology is never used, since Donald never activates it, hence it will disappear from the global type describing the composition of G_1 and G_2 using Bob and Carol as gateways:

$$\mathsf{G}_1 \ {}^{\mathsf{Bob}\mapsto\mathsf{Carol}} \ \mathsf{G}_2 = \\ \mathsf{Alice} \to \mathsf{Bob} : \mathit{req}.\mathsf{Bob} \to \mathsf{Carol} : \mathit{req}.\mathsf{Carol} \to \mathsf{Donald} : \mathit{req}. \\ \mathsf{Donald} \to \mathsf{Carol} : \mathit{adv}.\mathsf{Carol} \to \mathsf{Bob} : \mathit{adv}.\mathsf{Bob} \to \mathsf{Alice} : \mathit{adv}$$

Figure 2 shows the UML-like diagram corresponding to the above global type. In [1] it is shown that, by connecting systems of Communicating Finite State Machines (CFSMs) [9] as shown above, many relevant properties of communication, such as deadlock freedom, are preserved.

One of the contributions of the present paper is to provide a choreography formalism enabling the "lifting" of the composition-by-gateways approach in [1] from systems to protocol descriptions, represented as MPSTs. More precisely, we define a function that, given two MPSTs and two participants (one per MPST) to be used as interfaces, computes the MPST of the composed system, as informally shown above in the $\mathsf{G}_1 \ {}^{\mathsf{Bob}\mapsto\mathsf{Carol}} \ \mathsf{G}_2$ example. This allows one to lift to the composed system all the guarantees on the soundness of communication provided by the chosen MPST formalism.

In our formalism, in particular, typable systems are guaranteed to be lock-free [34]. Hence, the systems obtained by composing typable systems are lock-free too. Beyond this, the global type of the composed system also provides a global view of the new system. Both the benefits are direct consequences of one of our main contributions: the function from the global types of the original systems to the global type of the system obtained by the composition outlined above.

We remark that our approach does not change the syntax of the chosen MPST approach, but allows one to look at it as a formalism to describe open systems (while MPSTs normally describe only closed systems).

While the idea of the approach is rather general, not all MPST formalisms in the literature are suitable for its application. For instance, the requirements imposed by the type system in [19] are too strong for the gateway processes to be typed, thus we could not get any guarantee on the result of the composition.

In the present paper we apply the approach in a setting as simple as possible (inspired by [43]), to highlight its features and avoid non intrinsic complexity. In particular, the simplicity of the calculus allows us to get rid of channels and local types. Moreover, as discussed in Section 11, our treatment of infinitary computations avoids the hindering issues caused by a syntactic description of recursion.

In order to ensure the correctness of composition, we appeal to *interface compatibility*, a condition on the two participants chosen as interfaces. We show that the compatibility relation used in [1] can be relaxed to a relation closely connected to the observational preorder of [43], in turn corresponding to the subtyping relation for session types of [28, 22]. In particular, in [1] the interaction behaviour of Carol – in order to be compatible with Bob's one – should necessarily be Donald!req.Donald?$\{\mathit{adv}, \mathit{sorry}\}$ (or Bob's should be Alice?req.Alice!adv). We show instead that the notion of compatibility can actually be relaxed so to allow our Bob and Carol in (1) to be compatible.

The notion of compatibility between global types is further investigated by showing it to be, not only a sufficient condition to get lock-freedom preservation under composition, but also a necessary condition.

The use of gateways enables a safe connection of systems by means of a minimal modification, corresponding to the transformation of interface participants into forwarders, while the other participants are unchanged. This property is quite important when composition happens dynamically. However this approach is less suitable when composition is performed statically, to support a modular design and development process. Indeed, in this case one would like to avoid the overhead due to gateways.

For instance, in our simple example, the composition of System **1** and System **2** could be obtained by "bypassing" the interface partecipants, so getting an MPST whose UML-like diagram is shown in Figure 3.

To formalise such an idea, we define a second function which performs *direct composition* (both at the level of global types and at the level of systems),

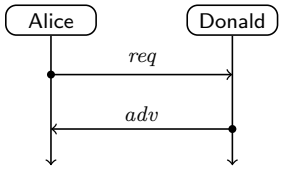


Figure 3: UML-like diagram of Systems **1** and **2** direct composition

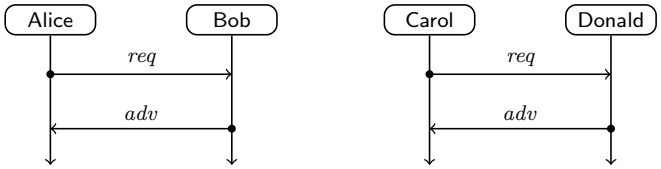


Figure 4: UML-like diagrams of the decomposition of the system of Fig. 3

that is composition not mediated by gateways. One can see direct composition as a simplification of composition by gateways, where the behaviour of gateways is internalised in the participants willing to communicate with the other system, hence gateways are no more needed. Of course, this approach requires some (limited) changes to the other participants of the systems to be composed.

Moreover, we complement direct composition with an operation of *decomposition*, which is, on global types, its left inverse. Decomposition allows one to take a typed MPST and split its participants into two groups (to each group an interface participant is also added), thus creating two MPSTs. A MPST with the original global type can be recovered by direct composition. For example the decomposition of the system with the UML-diagram of Figure 3 by adding Bob and Carol gives the systems represented in Figure 4. Direct composition and decomposition form the basis of a modular design and implementation development process, where one can take a global description, divide it into descriptions of subsystems to be implemented separately, and then safely compose the resulting implementations.

In the standard subtyping for session types supertypes have less inputs and more outputs than their subtypes [22]. In our type system we use a preorder on processes in which larger processes have less inputs than smaller ones, but the same outputs. This allows us to get: *(i)* a stronger notion of session fidelity, *(ii)* a better correspondence between composition by gate-

ways at the level of multiparty sessions and the one at the level of global types, *(iii)* the necessity of compatibility between global types to ensure that lock-freedom is preserved under composition. Besides, we show that such a restriction does not change the class of multiparty sessions that can be typed (but may change the types assigned to them).

Outline. Sections 2, 3 and 4 respectively introduce our calculus of multiparty sessions, their global types, and prove the properties that well-typed sessions enjoy. Sections 5 and 6 define the compatibility relation and the composition via gateways for sessions and global types, respectively. The key result that sessions obtained from composition are typable (and hence lock free) is presented in Theorem 6.10. In Section 7 compatibility between global types is shown to be not only a sufficient, but also a necessary condition for lock-freedom of composition. Direct composition, allowing one to compose systems without the need for gateways, is defined in Section 8, where related properties are also discussed and proved. The decomposition operation, which complements the direct composition operation, is defined and investigated in Section 9. In Section 10 we discuss how all previous results change if we consider a preorder on processes mimicking the standard subtyping relation on session types, that allows larger processes to have more outputs than smaller ones. Section 11 concludes by a recap of the paper and by discussing related and future work.

Comparison with the workshop version. The present paper is a revised and extended version of [3]. The definition of the composition operation, first presented in [3], is considerably improved here, enabling us to simplify and make more readable also the proofs of most of the related results. This paper includes also a number of new contributions. First, we show that compatibility between global types is not only a sufficient, but also a necessary condition in order to get lock-freedom preservation for composition of typable multiparty sessions. Also, both the direct composition and the decomposition operations, and all the related results, are new. Lastly, in the workshop version we did not consider how the results change when the processes are compared using a preorder mimicking the standard subtyping.

2. Processes and Multiparty Sessions

We use the following base sets and notation: *messages*, ranged over by $\ell, \ell', \ldots$; *session participants*, ranged over by $\mathsf{p}, \mathsf{q}, \ldots$; *processes*, ranged over

by $P, Q, \ldots$; *multiparty sessions*, ranged over by $\mathcal{M}, \mathcal{M}', \ldots$; *integers*, ranged over by $n, m, i, j, \ldots$.

Processes implement the behaviour of participants. The input process $\mathsf{p}?\{\ell_i.P_i \mid {}_{1\leq i\leq n}\}$ waits for one of the messages ℓ_i from participant p; the output process $\mathsf{p}!\{\ell_i.P_i \mid {}_{1\leq i\leq n}\}$ non-deterministically chooses one message ℓ_i for some i, $1 \leq i \leq n$, and sends it to participant p. We use Λ as shorthand for $\{\ell_i.P_i \mid {}_{1\leq i\leq n}\}$. We define the multiset of messages in Λ as $msg(\{\ell_i.P_i \mid {}_{1\leq i\leq n}\}) = \{\ell_i \mid {}_{1\leq i\leq n}\}$. After sending or receiving the message ℓ_i for some i, the process reduces to P_i. The set Λ acts as an *external choice* in $\mathsf{p}?\Lambda$ and as an *internal choice* in $\mathsf{p}!\Lambda$. In a full-fledged calculus, messages would carry values, namely they would be of the form $\ell(\mathsf{v})$. For simplicity, we consider only pure messages. This agrees with the focus of session calculi, which is on process interactions that do not depend on actual transmitted values.

For the sake of abstraction, we do not take into account any explicit syntax for recursion. Rather, we consider processes as, possibly infinite, regular trees.

Definition 2.1 (Processes). *Let P range over the terms of the following grammar:*

$$P \;::=^{coinductive}\; \mathbf{0} \;\mid\; \mathsf{p}?\Lambda \;\mid\; \mathsf{p}!\Lambda \qquad\qquad \Lambda \;::= \{\ell_i.P_i \mid {}_{1\leq i\leq n}\}$$

where all messages in $msg(\Lambda)$ are pairwise distinct. We call Λ a pre-choice *and define the* tree representation *of P as the directed rooted tree such that*

(a) each internal node is labelled by $\mathsf{p}?$ or $\mathsf{p}!$ and has as many children as the number of messages

(b) the edge from $\mathsf{p}?$ or $\mathsf{p}!$ to the child P_i is labelled by ℓ_i and

(c) the leaves of the tree (if any) are labelled by $\mathbf{0}$.

The term P is a process *if its tree representation is regular (namely, it has finitely many distinct sub-trees) and the pre-choice $\{\ell_i.P_i \mid {}_{1\leq i\leq n}\}$ is a* choice of messages *if P_i is a process for all $1 \leq i \leq n$.*

The fact that the grammar for P has to be interpreted coinductively implies that infinite derivations are allowed. We identify processes with their

tree representations and we shall sometimes refer to the trees as the processes themselves. The regularity condition implies that we only consider processes admitting a finite description. This is equivalent to writing processes with μ-notation and having an equality which allows for an infinite number of unfoldings. This is also called the *equirecursive approach*, since it views processes as the unique solutions of (guarded) recursive equations [41, Section 20.2]. The existence and uniqueness of a solution follow from known results (see [20] and [15, Theorem 7.5.34]). With such an approach, it is natural to use *coinduction* as the main logical tool, as we do in most of the proofs. In particular, we adopt the coinduction style advocated in [35] which, without compromising formal rigour, promotes readability and conciseness.

We define the set $\mathtt{ptp}(P)$ of participants of process P by:

$$\begin{array}{ll} \mathtt{ptp}(\mathbf{0}) & = \emptyset \\ \mathtt{ptp}(\mathtt{p}?\{\ell_i.P_i \mid 1{\leq}i{\leq}n\}) & = \mathtt{ptp}(\mathtt{p}!\{\ell_i.P_i \mid 1{\leq}i{\leq}n\}) = \{\mathtt{p}\}{\cup}\mathtt{ptp}(P_1){\cup}\ldots{\cup}\mathtt{ptp}(P_n) \end{array}$$

The regularity of processes assures that the set of participants is finite.
We shall write $\ell.P \uplus \Lambda$ for $\{\ell.P\}\cup\Lambda$ if $\ell \notin msg(\Lambda)$, and $\Lambda_1 \uplus \Lambda_2$ for $\Lambda_1 \cup \Lambda_2$ if $msg(\Lambda_1)\cap msg(\Lambda_2) = \emptyset$. We shall also omit curly brackets in choices with only one branch, as well as trailing $\mathbf{0}$ processes.

A *multiparty session* is the parallel composition of *named* processes, that is pairs participant/process.

Definition 2.2 (Multiparty Sessions). *A term $\mathcal{M}$ generated by the grammar*

$$\mathcal{M} ::=^{inductive} \mathtt{p} \triangleright P \;\mid\; \mathcal{M} \mid \mathcal{M}$$

is a multiparty session *if the following conditions hold:*

(a) in $\mathtt{p}_1 \triangleright P_1 \mid \ldots \mid \mathtt{p}_n \triangleright P_n$, for all i, j, $1 \leq i \neq j \leq n$, $\mathtt{p}_i \neq \mathtt{p}_j$;

(b) in $\mathtt{p} \triangleright P$ we require $\mathtt{p} \notin \mathtt{ptp}(P)$.

As we will see, our typing discipline makes condition (b) in the above definition redundant, since our global types forbid self-communications.

Hereafter, we use $\prod_{1\leq i\leq n} \mathtt{p}_i \triangleright P_i$ as shorthand for $\mathtt{p}_1 \triangleright P_1 \mid \ldots \mid \mathtt{p}_n \triangleright P_n$ and we write $\mathtt{p}_j \triangleright P_j \in \mathcal{M}$ when $\mathcal{M} = \prod_{1\leq i\leq n} \mathtt{p}_i \triangleright P_i$ and $1 \leq j \leq n$.

We define $\mathtt{pts}(\mathtt{p} \triangleright P) = \{\mathtt{p}\}$ and $\mathtt{pts}(\mathcal{M} \mid \mathcal{M}') = \mathtt{pts}(\mathcal{M}) \cup \mathtt{pts}(\mathcal{M}')$.

Operational Semantics. We assume to have a structural congruence $\equiv$ on multiparty sessions, establishing that parallel composition is commutative, associative and has neutral elements $\mathsf{p} \triangleright \mathbf{0}$ for any fresh p.

The reduction for multiparty sessions allows participants to choose and communicate messages.

Definition 2.3 (LTS for Multiparty Sessions). *The* labelled transition system (LTS) for multiparty sessions *is the closure under structural congruence of the reduction specified by the unique rule:*

$$[\textsc{comm}] \quad \frac{msg(\Lambda) \subseteq msg(\Lambda')}{\mathsf{p} \triangleright \mathsf{q}!(\ell.P \uplus \Lambda) \;\mid\; \mathsf{q} \triangleright \mathsf{p}?(\ell.Q \uplus \Lambda') \mid \mathcal{M} \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathsf{p} \triangleright P \;\mid\; \mathsf{q} \triangleright Q \mid \mathcal{M}}$$

Rule [comm] makes the communication possible: participant p sends message ℓ to participant q. This rule is non-deterministic in the choice of messages. The condition $msg(\Lambda) \subseteq msg(\Lambda')$ (borrowed from [4, 5]) assures 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 and external choices. Note that this condition will always be true in well-typed sessions. For example, $\mathsf{p} \triangleright \mathsf{q}!\ell \mid \mathsf{q} \triangleright \mathsf{p}?\{\ell, \ell'\} \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathsf{p} \triangleright \mathbf{0} \mid \mathsf{q} \triangleright \mathbf{0}$, while $\mathsf{p} \triangleright \mathsf{q}!\{\ell, \ell'\} \mid \mathsf{q} \triangleright \mathsf{p}?\ell$ is stuck, since participant p should be free of choosing to send message ℓ' to participant q (p is making an internal choice), but q cannot receive this message.

We use $\mathcal{M} \xrightarrow{\lambda} \mathcal{M}'$ as shorthand for $\mathcal{M} \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathcal{M}'$ for some p, q and ℓ. We sometimes omit the label, writing $\longrightarrow$. As usual, $\longrightarrow^*$ denotes the reflexive and transitive closure of $\longrightarrow$.

Example 2.4. Let us consider a system (inspired by a similar one in [1]) with participants p, q, and h interacting according to the following protocol. Participant p keeps on sending text messages to q, which has to deliver them to h. After a message has been sent by p, the next one can be sent only if h has ascertained the propriety of language of the previous message, i.e if it does not contain, say, rude or offensive words. Participant h acknowledges to q the propriety of language of a received text by means of the message *ack*. In such a case q sends to p an *ok* message so that p can proceed by sending a further message. More precisely:

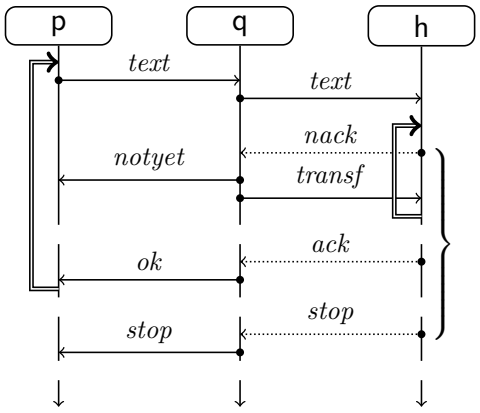


Figure 5: UML-like diagram for Example 2.4

1. p sends a text message to q in order to be delivered to h, which accepts only texts possessing a good propriety of language;

2. then h either

   (a) sends an *ack* to q certifying the reception of the text and its propriety. In this case q sends back to p an *ok* message and the protocol goes back to step 1, so that p can proceed by sending a further text message;

   (b) sends a *nack* message to inform q that the text has not the required propriety of language. In such a case q produces *transf* (a semantically invariant reformulation of the text), sends it back to h and the protocol goes to step 2 again. Before doing that, q informs p (through the *notyet* message) that the text has not been accepted yet and a reformulation has been requested;

   (c) sends a *stop* message to inform q that no more text will be accepted. In such a case q informs p of that.

The above protocol can be described by the UML-like diagram of Figure 5. There, doubly-lined arrows denote iteration. The first communication of each branch in a choice is dotted, and the description of the branch ends where the vertical lines break. For graphical reasons the alternatives *ack* and *nack* appear in reverse order.

The multiparty session $\mathcal{M} = \mathsf{p} \triangleright P \mid \mathsf{q} \triangleright Q \mid \mathsf{h} \triangleright H$ where

$$
\begin{array}{ll}
P = \mathsf{q}!\mathit{text}.P_1 & P_1 = \mathsf{q}?\{\mathit{ok}.P, \mathit{notyet}.P_1, \mathit{stop}\} \\
Q = \mathsf{p}?\mathit{text}.\mathsf{h}!\mathit{text}.Q_1 & Q_1 = \mathsf{h}?\{\mathit{ack}.\mathsf{p}!\mathit{ok}.Q, \\
 & \qquad \mathit{nack}.\mathsf{p}!\mathit{notyet}.\mathsf{h}!\mathit{transf}.Q_1, \\
 & \qquad \mathit{stop}.\mathsf{p}!\mathit{stop}\} \\
H = \mathsf{q}?\mathit{text}.H_1 & H_1 = \mathsf{q}!\{\mathit{ack}.H, \mathit{nack}.\mathsf{q}?\mathit{transf}.H_1, \mathit{stop}\}
\end{array}
$$

implements the protocol in Figure 5. ◇

We end this section by defining the property of lock-freedom for multiparty sessions as in [34, 40]. Lock-freedom guarantees progress for each participant (and hence deadlock-freedom). In other words, each participant ready to communicate will eventually find her partner exposing a dual communication action.

Definition 2.5 (Deadlock-freedom and Lock-freedom). *Let $\mathcal{M}$ be a multiparty session.*

(i) $\mathcal{M}$ is a deadlock-free session *if* $\mathcal{M} \longrightarrow^* \mathcal{M}'$ *implies either* $\mathcal{M}' \equiv \mathsf{p} \triangleright \mathbf{0}$ *or* $\mathcal{M}' \longrightarrow \mathcal{M}''$*;*

(ii) $\mathcal{M}$ is a lock-free session *if* $\mathcal{M} \longrightarrow^* \mathsf{p} \triangleright P \mid \mathcal{M}'$ *and* $P \neq \mathbf{0}$ *imply* $\mathsf{p} \triangleright P \mid \mathcal{M}' \longrightarrow^* \mathcal{M}'' \xrightarrow{\lambda}$ *where* p *occurs in* λ*.*

Recall that $\mathsf{p} \triangleright \mathbf{0}$ is a neutral element of $\mid$ for the structural equivalence $\equiv$ of multiparty sessions, and then $\mathsf{p} \triangleright \mathbf{0} \mid \prod_{1 \leq i \leq n} \mathsf{p}_i \triangleright \mathbf{0} \equiv \mathsf{p} \triangleright \mathbf{0}$.

It can be easily proved that lock-freedom implies deadlock-freedom, but not vice versa.

Proposition 2.6.
Lock-freedom implies deadlock-freedom, but not vice versa.

Proof. Let us first show that lock-freedom implies deadlock-freedom. Let $\mathcal{M}$ be lock-free. In order to show it to be also deadlock-free, let $\mathcal{M} \longrightarrow^* \mathcal{M}'$. We consider the two possible cases below:

$\mathcal{M}' \equiv \mathsf{p} \triangleright \mathbf{0}$
Deadlock-freedom descends hence immediately by Definition 2.5(i).

$\mathcal{M}' \not\equiv \mathsf{p} \triangleright \mathbf{0}$

In such a case we have necessarily that $\mathcal{M}' \equiv \mathsf{p} \triangleright P \mid \mathcal{M}''$ (and hence $\mathcal{M}' \longrightarrow^* \mathsf{p} \triangleright P \mid \mathcal{M}''$) for some p, $\mathcal{M}''$ and $P \neq \mathbf{0}$. By lock-freedom, it follows that $\mathsf{p} \triangleright P \mid \mathcal{M}' \longrightarrow^* \mathcal{M}'' \xrightarrow{\lambda}$ with p occurring in λ. Then, trivially, $\mathcal{M}' \longrightarrow \mathcal{M}'''$ for some $\mathcal{M}'''$, and we are done.

In order to show that the opposite implication does not hold, let us consider the following simple multiparty session:

$$\mathcal{M} = \mathsf{p} \triangleright P \mid \mathsf{q} \triangleright Q \mid \mathsf{r} \triangleright \mathsf{s}!\ell$$

where $P = \mathsf{q}!\ell'.P$ and $Q = \mathsf{p}?\ell'.Q$.
It is not difficult to check that $\mathcal{M}$ is deadlock-free, whereas, since the process of r cannot communicate at all, $\mathcal{M}$ is not lock-free. □

To get a stronger lock-freedom assuring that a communication involving p will occur in all reduction sequences starting from $\mathsf{p} \triangleright P \mid \mathcal{M}$ when $P \neq \mathbf{0}$ we need the following fairness assumption:

> *in a session, any pair of participants ready to communicate will exchange a message after a finite number of steps.*

In this way the multiparty session $\mathcal{M} = \mathsf{p} \triangleright P \mid \mathsf{q} \triangleright Q \mid \mathsf{r} \triangleright \mathsf{s}!\ell \mid \mathsf{s} \triangleright \mathsf{r}?\ell$ with $P = \mathsf{q}!\ell'.P$ and $Q = \mathsf{p}?\ell'.Q$ will not have the infinite reduction

$$\mathcal{M} \xrightarrow{\mathsf{p}\ell'\mathsf{q}} \mathcal{M} \xrightarrow{\mathsf{p}\ell'\mathsf{q}} \cdots$$

where r and s never communicate.
This assumption can be implemented, e.g., by a maximally parallel reduction in which at each step all possible communications are done [43].
Notice that we do not require participants to choose outputs in a fair way, as exemplified below. This is not needed, since our type system will avoid starvation due to the choice of labels.

Example 2.7. The multiparty session $\mathcal{M} = \mathsf{p} \triangleright P \mid \mathsf{q} \triangleright Q \mid \mathsf{r} \triangleright R$ with $P = \mathsf{q}!\{\ell_1.\mathsf{r}?\ell_3, \ell_2.P\}$, $Q = \mathsf{p}?\{\ell_1, \ell_2.Q\}$, and $R = \mathsf{p}!\ell_3$, has the infinite reduction

$$\mathcal{M} \xrightarrow{\mathsf{p}\ell_2\mathsf{q}} \mathcal{M} \xrightarrow{\mathsf{p}\ell_2\mathsf{q}} \cdots$$

in which participant r can never send her message. This multiparty session will be rejected by our type system, see Remark 3.11. ◇

3. Global Types and Typing System

The behaviour of multiparty sessions can be disciplined by means of types. Global types describe the whole conversation scenarios of multiparty sessions. As in [43] we directly assign global types to multiparty sessions without the usual detour around session types and subtyping [31, 32].

The type $\mathsf{p} \to \mathsf{q} : \{\ell_i.\mathsf{G}_i \mid {}_{1\leq i\leq n}\}$ formalises a protocol where participant p must send to q a message ℓ_i for some i, $1 \leq i \leq n$, (and q must receive it) and then, depending on which ℓ_i was chosen by p, the protocol continues as G_i. We use Γ as shorthand for $\{\ell_i.\mathsf{G}_i \mid {}_{1\leq i\leq n}\}$ and define the multiset $msg(\{\ell_i.\mathsf{G}_i \mid {}_{1\leq i\leq n}\}) = \{\ell_i \mid 1 \leq i \leq n\}$.

Definition 3.1 (Global Types). *Let* G *range over the terms of the following grammar:*

$$\mathsf{G} ::=^{coinductive} \ \mathtt{End} \ \mid \ \mathsf{p} \to \mathsf{q} : \Gamma \qquad \Gamma := \{\ell_i.\mathsf{G}_i \mid {}_{1\leq i\leq n}\}$$

where $\mathsf{p} \neq \mathsf{q}$ *and all messages in* $msg(\Gamma)$ *are pairwise distinct. We call* Γ *a* pre-choice of communications *and define the* tree representation *of* G *as the directed rooted tree such that*

(a) each internal node is labelled by $\mathsf{p} \to \mathsf{q}$ *and has as many children as the number of messages,*

(b) the edge from $\mathsf{p} \to \mathsf{q}$ *to the child* G_i *is labelled by* ℓ_i *and*

(c) the leaves of the tree (if any) are labelled by $\mathtt{End}$.

The term G *is a* global type *if its tree representation is regular (namely, it has finitely many distinct sub-trees) and the pre-choice* $\{\ell_i.\mathsf{G}_i \mid {}_{1\leq i\leq n}\}$ *is a* choice of communications *if* G_i *is a global type for all* $i, 1 \leq i \leq n$.

We identify global types with their tree representations and we shall sometimes refer to the tree representations as the global types themselves. As for processes, the regularity condition implies that we only consider global types admitting a finite representation.

The set $\mathtt{ptg}(\mathsf{G})$ of participants of global type G is defined similarly to those of processes and sessions. The regularity of global types assures that the set of participants is finite. We shall write $\ell.\mathsf{G} \uplus \Gamma$ for $\{\ell.\mathsf{G}\} \cup \Gamma$ if $\ell \notin msg(\Gamma)$ and $\Gamma_1 \uplus \Gamma_2$ for $\Gamma_1 \cup \Gamma_2$ if $msg(\Gamma_1) \cap msg(\Gamma_2) = \emptyset$. We assume that $\uplus$ has the highest precedence, i.e. $\mathsf{p} \to \mathsf{q} : \Gamma_1 \uplus \Gamma_2$ means $\mathsf{p} \to \mathsf{q} : (\Gamma_1 \uplus \Gamma_2)$. We

shall omit curly brackets in choices with only one branch, as well as trailing `End` terms.

Since all messages in communication choices are pairwise distinct, the set of paths in the trees representing global types are determined by the labels of nodes and edges found on the way. Let ρ range over paths of global types. Formally the set of paths of a global type can be defined as a set of sequences (ϵ is the empty sequence):

$$\begin{array}{lcl} \mathit{paths}(\mathtt{End}) & = & \{\epsilon\} \\ \mathit{paths}(\mathsf{p} \to \mathsf{q} : \{\ell_i.\mathsf{G}_i \mid {}_{1\leq i\leq n}\}) & = & \bigcup_{1\leq i\leq n}\{(\mathsf{p} \to \mathsf{q})\,\ell_i\,\rho \mid \rho \in \mathit{paths}(\mathsf{G}_i)\} \end{array}$$

Note that every infinite path of a global type has infinitely many occurrences of '$\to$'. We use "$\mathsf{p} \notin \rho$" as a shorthand for "p *does not occur in* ρ".
The function *length* returns the number of "interactions" in a path, and is defined as expected for finite paths:

$$\begin{array}{lcl} \mathit{length}(\epsilon) & = & 0 \\ \mathit{length}((\mathsf{p} \to \mathsf{q})\,\ell_i\,\rho) & = & 1 + \mathit{length}(\rho) \end{array}$$

whereas we set $\mathit{length}(\rho) = \infty$ for infinite paths.

Example 3.2. A global type representing the protocol of Example 2.4 is:

$$\begin{array}{lcl} \mathsf{G} & = & \mathsf{p} \to \mathsf{q} : \mathit{text}.\mathsf{q} \to \mathsf{h} : \mathit{text}.\mathsf{G}_1 \\ \mathsf{G}_1 & = & \mathsf{h} \to \mathsf{q} : \{\mathit{ack}.\mathsf{q} \to \mathsf{p} : \mathit{ok}.\mathsf{G}, \\ & & \quad \mathit{nack}.\mathsf{q} \to \mathsf{p} : \mathit{notyet}.\mathsf{q} \to \mathsf{h} : \mathit{transf}.\mathsf{G}_1, \\ & & \quad \mathit{stop}.\mathsf{q} \to \mathsf{p} : \mathit{stop}\} \end{array}$$

Notice that G faithfully corresponds to the diagram in Figure 5.

Examples of paths in G are

$$\begin{array}{ll} \rho_1 = (\mathsf{p} \to \mathsf{q})\mathit{text}(\mathsf{q} \to \mathsf{h})\mathit{text}(\mathsf{h} \to \mathsf{q})\mathit{stop}(\mathsf{q} \to \mathsf{p})\mathit{stop} & \text{and} \\ \rho_2 = (\mathsf{p} \to \mathsf{q})\mathit{text}(\mathsf{q} \to \mathsf{h})\mathit{text}(\mathsf{h} \to \mathsf{q})\mathit{ack}(\mathsf{q} \to \mathsf{p})\mathit{ok}\rho_2 & \end{array}$$

We get $\mathit{length}(\rho_1) = 4$ and $\mathit{length}(\rho_2) = \infty$. ◇

Usually, as in [31, 32], projection of global types onto participants produces session types, and session types are assigned to processes by a type system. The simple shape of our messages, instead, allows us to define a projection of global types onto participants producing processes directly.

The projection of a global type onto a participant, if defined, returns the process that the participant should run to comply with the protocol specified by the global type. If the global type begins with a message from p to q, then the projection onto p should send one message to q, and the projection onto q should receive one message from p. The projection onto a third participant r skips the initial communication, that does not involve r. However, such a communication may be part of a choice, and since r is not part of it, she is not immediately aware of which branch is taken. Then, there are two possibilities: either r will behave in the same way in all the branches, hence she has no need to know which one has been taken, or she will discover later on which branch has been taken, by receiving distinct messages in each branch. In the first case, the projections on r of all the branches must be the same, and this projection defines her behaviour. In the second case, the projections yield input processes receiving from the same sender, and we can allow the process of r to combine all these processes, on the proviso the messages are all different.

Definition 3.3 (Projection). *Given a participant* p*, we coinductively define the partial function* $\_\upharpoonright_{\mathsf{p}}$ *on global types* G *as follows:*

$$
\begin{array}{lcl}
\mathsf{G}\upharpoonright_{\mathsf{p}} & = & \mathbf{0} \quad \textit{if } \mathsf{p} \notin \mathtt{ptg}(\mathsf{G}) \\
(\mathsf{p} \to \mathsf{q} : \{\ell_i.\mathsf{G}_i \mid {\scriptstyle 1 \le i \le n}\})\upharpoonright_{\mathsf{p}} & = & \mathsf{q}!\{\ell_i.\mathsf{G}_i\upharpoonright_{\mathsf{p}} \mid {\scriptstyle 1 \le i \le n}\} \\
(\mathsf{q} \to \mathsf{p} : \{\ell_i.\mathsf{G}_i \mid {\scriptstyle 1 \le i \le n}\})\upharpoonright_{\mathsf{p}} & = & \mathsf{q}?\{\ell_i.\mathsf{G}_i\upharpoonright_{\mathsf{p}} \mid {\scriptstyle 1 \le i \le n}\} \\
(\mathsf{q} \to \mathsf{r} : \{\ell_i.\mathsf{G}_i \mid {\scriptstyle 1 \le i \le n}\})\upharpoonright_{\mathsf{p}} & = & \\
\qquad = \begin{cases} \mathsf{G}_1\upharpoonright_{\mathsf{p}} & \textit{if } \mathsf{p} \notin \{\mathsf{q},\mathsf{r}\} \textit{ and } \mathsf{G}_i\upharpoonright_{\mathsf{p}} = \mathsf{G}_1\upharpoonright_{\mathsf{p}} \ \ \forall\, 1 \le i \le n \\ \mathsf{s}?(\Lambda_1 \uplus \ldots \uplus \Lambda_n) & \textit{if } \mathsf{p} \notin \{\mathsf{q},\mathsf{r}\} \textit{ and } \mathsf{G}_i\upharpoonright_{\mathsf{p}} = \mathsf{s}?\Lambda_i \ \forall\, 1 \le i \le n \end{cases}
\end{array}
$$

We say that $\mathsf{G}\upharpoonright_{\mathsf{p}}$ *is the* projection *of* G *onto* p *if* $\mathsf{G}\upharpoonright_{\mathsf{p}}$ *is defined. We say that* G *is* projectable *if* $\mathsf{G}\upharpoonright_{\mathsf{p}}$ *is defined for all participants* p.

This projection is the coinductive version of the projection given in [23, 29], where processes are replaced by local types.

As mentioned above, if p is not involved in the first communication of G, then in all branches the process of participant p must either behave in the same way or be an input from the same sender of different messages, so that p can understand which branch was chosen.

Example 3.4. The global type G of Example 3.2 is projectable, and by projecting it we obtain $\mathsf{G}\upharpoonright_{\mathsf{p}} = P$, $\mathsf{G}\upharpoonright_{\mathsf{q}} = Q$, $\mathsf{G}\upharpoonright_{\mathsf{h}} = H$, where P, Q, and H are

as defined in Example 2.4.
Also the global type $\mathsf{G}' = \mathsf{p} \to \mathsf{q} : \{\ell_1.\mathsf{r} \to \mathsf{p} : \ell_3, \ell_2.\mathsf{G}'\}$ is projectable: $\mathsf{G}'\upharpoonright_{\mathsf{p}} = \mathsf{q}!\{\ell_1.\mathsf{r}?\ell_3, \ell_2.P\}$, $\mathsf{G}'\upharpoonright_{\mathsf{q}} = \mathsf{p}?\{\ell_1, \ell_2.Q\}$, $\mathsf{G}'\upharpoonright_{\mathsf{r}} = \mathsf{p}!\ell_3$. These are the processes of the multiparty session shown in Example 2.7. Notice that G' has two branches, the projection of the first branch onto r is $\mathsf{p}!\ell_3$, the projection of the second branch onto r is just the projection of G' onto r, so $\mathsf{p}!\ell_3$ is the (coinductive) projection of G' onto r. $\diamond$

In order to assure lock-freedom by typing, we require each participant to occur in all the paths from the root, and that for each participant the first occurrences in the various paths are at bounded depth. This second constraint is formalised by requiring the *depth* of each participant in a global type – defined below in terms of the *depth* of a participant in a path – to be finite.

Definition 3.5 (Depth). *Let*

$$depth(\rho, \mathsf{p}) = \begin{cases} length(\rho_1) & \text{if } \rho = \rho_1\,(\mathsf{q} \to \mathsf{r})\,\ell\,\rho_2,\ \mathsf{p} \notin \rho_1 \text{ and } \mathsf{p} \in \{\mathsf{q}, \mathsf{r}\} \\ 0 & \text{if } \mathsf{p} \notin \rho \end{cases}$$

We then define $depth : \mathsf{G} \times \mathtt{ptg}(\mathsf{G}) \to \mathbb{N} \cup \{\infty\}$ *by*

$$depth(\mathsf{G}, \mathsf{p}) = sup\{depth(\rho, \mathsf{p}) \mid \rho \in paths(\mathsf{G})\}$$

Example 3.6. Let G be as in Example 3.2; then $depth(\mathsf{G}, \mathsf{p}) = depth(\mathsf{G}, \mathsf{q}) = 0$, and $depth(\mathsf{G}, \mathsf{h}) = 1$. Let $\mathsf{G}' = \mathsf{p} \to \mathsf{q} : \{\ell_1.\mathsf{r} \to \mathsf{p} : \ell_3, \ell_2.\mathsf{G}'\}$ as in Example 3.4, then $depth(\mathsf{G}', \mathsf{r}) = \infty$. $\diamond$

Definition 3.7 (Well-formed Global Types). *A global type* G *is* well-formed *if:*

(a) $depth(\mathsf{G}', \mathsf{p})$ *is finite for all* G' *subterms of* G *and all* $\mathsf{p} \in \mathtt{ptg}(\mathsf{G})$*;*

(b) $\mathsf{G}\upharpoonright_{\mathsf{p}}$ *is defined for all* $\mathsf{p} \in \mathtt{ptg}(\mathsf{G})$*.*

Example 3.8. The global type G of Example 3.2 is well-formed, while the global type G' of Example 3.6 is not well-formed, since the depth of r in G' is not bounded.

The necessity of considering the depth of subterms in the definition of well-formedness is justified by the example

$$\mathsf{G}'' = \mathsf{p} \to \mathsf{r} : \ell_0.\mathsf{p} \to \mathsf{q} : \{\ell_1.\mathsf{r} \to \mathsf{p} : \ell_3, \ell_2.\mathsf{G}'\}$$

where G' is as above. In fact $\mathit{depth}(\mathsf{G}'', \mathsf{r}) = 0$, while $\mathit{depth}(\mathsf{G}', \mathsf{r}) = \infty$. Therefore G'' is not well-formed, and this is important as we will show in Example 4.4. $\diamond$

In the following, **we will only consider well-formed global types**.

To type multiparty sessions we use the preorder $\leqslant$ on processes defined below and inspired by the subtyping of [17].

Definition 3.9 (Structural Preorder). *We define the* structural preorder on processes*, $P \leqslant Q$, by coinduction:*

$$[\textsc{sub-}\mathbf{0}] \quad \mathbf{0} \leqslant \mathbf{0} \qquad \frac{[\textsc{sub-out}] \quad P_i \leqslant Q_i \quad \forall 1 \leq i \leq n}{\mathsf{p}!\{\ell_i.P_i \mid {\scriptstyle 1 \leq i \leq n}\} \leqslant \mathsf{p}!\{\ell_i.Q_i \mid {\scriptstyle 1 \leq i \leq n}\}}$$

$$\frac{[\textsc{sub-in}] \quad P_i \leqslant Q_i \quad \forall 1 \leq i \leq n}{\mathsf{p}?(\{\ell_i.P_i \mid {\scriptstyle 1 \leq i \leq n}\} \uplus \Lambda) \leqslant \mathsf{p}?\{\ell_i.Q_i \mid {\scriptstyle 1 \leq i \leq n}\}}$$

The double-line in rules indicates that the rules are interpreted *coinductively*. Rule [sub-in] allows larger processes to offer fewer inputs than smaller ones, while Rule [sub-out] requires the output messages to be the same. The regularity condition on processes is crucial to guarantee the termination of algorithms for checking structural preorder. Usually, subtyping relations for output allow more branches in the supertype. We discuss this option in Section 10.

The typing judgments associate global types to sessions and are of the shape $\vdash \mathcal{M} : \mathsf{G}$.

Definition 3.10 (Typing System). *The only typing rule is:*

$$[\textsc{t-sess}] \quad \frac{\mathsf{p} \triangleright P \in \mathcal{M} \implies P \leqslant \mathsf{G}\restriction_{\mathsf{p}} \qquad \mathtt{ptg}(\mathsf{G}) \subseteq \mathtt{pts}(\mathcal{M})}{\vdash \mathcal{M} : \mathsf{G}}$$

A session $\mathcal{M}$ is well-typed *if there exists G such that $\vdash \mathcal{M} : \mathsf{G}$.*

The rule above requires that the processes in parallel can play as participants of a whole communication protocol or they are the terminated process, i.e. they are smaller or equal (according to the structural preorder) to the projections of a same global type. Note that [T-SESS] is not coinductive since a multiparty session is a finite parallel composition of (possibly infinite) processes; hence coinduction is necessary only for the subtyping relation. In general, $\mathtt{pts}(\mathcal{M}) \subseteq \mathtt{ptg}(\mathsf{G})$ does not hold, for example

$$\vdash \mathsf{p} \triangleright \mathsf{q}!\ell \mid \mathsf{q} \triangleright \mathsf{p}?\ell \mid \mathsf{r} \triangleright \mathbf{0} : \mathsf{p} \to \mathsf{q} : \ell$$

Remark 3.11. Since the projections of $\mathsf{G}' = \mathsf{p} \to \mathsf{q} : \{\ell_1.\mathsf{r} \to \mathsf{p} : \ell_3, \ell_2.\mathsf{G}'\}$ as defined in Example 3.4 are exactly the processes $P = \mathsf{q}!\{\ell_1.\mathsf{r}?\ell_3, \ell_2.P\}$, $Q = \mathsf{p}?\{\ell_1, \ell_2.Q\}$, and $R = \mathsf{p}!\ell_3$ of the multiparty session in Example 2.7 and G' is not well-formed, our type system rejects the multiparty session in Example 2.7.

4. Properties of Well-Typed Sessions

We start with the standard lemmas of inversion and canonical form, easily following from Rule [T-SESS].

Lemma 4.1 (Inversion Lemma). *If* $\vdash \mathcal{M} : \mathsf{G}$ *and* $\mathsf{p} \triangleright P \in \mathcal{M}$*, then*

$$P \leqslant \mathsf{G}\upharpoonright_{\mathsf{p}} .$$

Lemma 4.2 (Canonical Form Lemma). *If* $\vdash \mathcal{M} : \mathsf{G}$ *and* $\mathsf{p} \in \mathtt{ptg}(\mathsf{G})$*, then there is* $\mathsf{p} \triangleright P \in \mathcal{M}$ *and* $P \leqslant \mathsf{G}\upharpoonright_{\mathsf{p}}$.

To formalise the properties of Subject Reduction and Session Fidelity [31, 32], we use the standard LTS for global types given below.

Definition 4.3 (LTS for Global Types). *The* labelled transition system (LTS) for global types *is specified by the rules:*

$$[\text{ECOMM}] \quad \mathsf{p} \to \mathsf{q} : \ell.\mathsf{G} \uplus \Gamma \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathsf{G}$$

$$[\text{ICOMM}] \quad \frac{\mathsf{G}_i \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathsf{G}_i' \quad \forall 1 \leq i \leq n \qquad \{\mathsf{p},\mathsf{q}\} \cap \{\mathsf{r},\mathsf{s}\} = \emptyset}{\mathsf{r} \to \mathsf{s} : \{\ell_i.\mathsf{G}_i \mid {\scriptstyle 1 \leq i \leq n}\} \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathsf{r} \to \mathsf{s} : \{\ell_i.\mathsf{G}_i' \mid {\scriptstyle 1 \leq i \leq n}\}}$$

Rule [Icomm] makes sense since, in a projectable global type $\mathsf{r} \to \mathsf{s} : \Gamma$, behaviours involving participants p and q ready to interact with each other uniformly in all branches can do so if they are both not involved in a previous interaction between r and s. In fact, the interaction between p and q is independent of the choice of r, and may be executed before it.

Example 4.4. Let $\mathsf{G}'' = \mathsf{p} \to \mathsf{r} : \ell_0.\mathsf{p} \to \mathsf{q} : \{\ell_1.\mathsf{r} \to \mathsf{p} : \ell_3, \ell_2.\mathsf{G}'\}$ and $\mathsf{G}' = \mathsf{p} \to \mathsf{q} : \{\ell_1.\mathsf{r} \to \mathsf{p} : \ell_3, \ell_2.\mathsf{G}'\}$ as in Example 3.8. We get $\mathsf{G}'' \xrightarrow{\mathsf{p}\ell\mathsf{r}} \mathsf{G}'$. Notice that $\mathit{depth}(\mathsf{G}'', \mathsf{r}) = 0$ and $\mathit{depth}(\mathsf{G}', \mathsf{r}) = \infty$. This shows that the finiteness of depth would not be preserved by reduction if we would not consider all subterms in condition (a) of Definition 3.7. $\diamond$

In the remainder of this section we show the main properties of our type system, i.e. Subject Reduction, Session Fidelity and Lock-Freedom. We start with three lemmas. The first lemma assures that well-formedness is preserved by the reduction of global types. The second lemma says that the depths of participants not occurring in the root of a global type decrease along the branches of the tree. The third lemma relates projections and reductions of global types.

Lemma 4.5. *If* $\mathsf{G} \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathsf{G}'$ *and* G *is well-formed, then* G' *is well-formed too.*

Proof. It is enough to observe that G' is a subterm of G. $\square$

Lemma 4.6. *If* $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \Gamma$ *and* $\mathit{depth}(\mathsf{G}, \mathsf{r}) \neq 0$*, then* $\mathit{depth}(\mathsf{G}, \mathsf{r}) > \mathit{depth}(\mathsf{G}', \mathsf{r})$ *for all* $\ell.\mathsf{G}' \in \Gamma$*.*

Proof. All paths of G are of the shape $(\mathsf{p} \to \mathsf{q})\,\ell\,\rho$, where ρ is a path of G'. This gives $\mathit{depth}(\mathsf{G}, \mathsf{r}) > \mathit{depth}(\mathsf{G}', \mathsf{r})$, since $\mathit{depth}(\mathsf{G}, \mathsf{r}) \neq 0$ by hypothesis and it is finite since G is well-formed. $\square$

Lemma 4.7 (Key Lemma). *(i) If* $\mathsf{G}\upharpoonright_{\mathsf{p}} = \mathsf{q}!\Lambda$ *and* $\mathsf{G}\upharpoonright_{\mathsf{q}} = \mathsf{p}?\Lambda'$*, then* $\mathit{msg}(\Lambda) = \mathit{msg}(\Lambda')$*. Moreover for all* $\ell \in \mathit{msg}(\Lambda)$*,* $\mathsf{G} \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathsf{G}^{\ell}$ *and* $\ell.\mathsf{G}^{\ell}\upharpoonright_{\mathsf{p}} \in \Lambda$ *and* $\ell.\mathsf{G}^{\ell}\upharpoonright_{\mathsf{q}} \in \Lambda'$*.*

(ii) If $\mathsf{G} \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathsf{G}'$*, then* $\mathsf{G}\upharpoonright_{\mathsf{p}} = \mathsf{q}!\Lambda$ *and* $\mathsf{G}\upharpoonright_{\mathsf{q}} = \mathsf{p}?\Lambda'$ *and* $\ell \in \mathit{msg}(\Lambda) = \mathit{msg}(\Lambda')$*.*

Proof. (i). The proof is by induction on $w = depth(\mathsf{G}, \mathsf{p})$.
If $w = 0$, then $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \Gamma$. Hence, by definition of projection (Definition 3.3), we have $msg(\Lambda) = msg(\Lambda') = msg(\Gamma)$ and, for all $\ell.\mathsf{G}^\ell \in \Gamma$, $\ell.\mathsf{G}^\ell\upharpoonright_\mathsf{p} \in \Lambda$ and $\ell.\mathsf{G}^\ell\upharpoonright_\mathsf{q} \in \Lambda'$. Moreover $\mathsf{G} \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathsf{G}^\ell$ using rule [ECOMM].
If $w > 0$, then $\mathsf{G} = \mathsf{r} \to \mathsf{s} : \{\ell_i.\mathsf{G}_i \mid 1 \leq i \leq n\}$ and $\mathsf{p} \notin \{\mathsf{r}, \mathsf{s}\}$. From $\mathsf{G}\upharpoonright_\mathsf{q} = \mathsf{p}?\Lambda'$ we get $\mathsf{q} \notin \{\mathsf{r}, \mathsf{s}\}$. Moreover $\mathsf{G}_i\upharpoonright_\mathsf{p} = \mathsf{q}!\Lambda$ and $\mathsf{G}_i\upharpoonright_\mathsf{q} = \mathsf{p}?\Lambda'$ for all i, $1 \leq i \leq n$, by definition of projection. By Lemma 4.6 $w > depth(\mathsf{G}_i, \mathsf{p})$ for all i, $1 \leq i \leq n$. By the induction hypothesis, $msg(\Lambda) = msg(\Lambda')$. Again by the induction hypothesis, $\mathsf{G}_i \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathsf{G}_i^\ell$ and $\ell.\mathsf{G}_i^\ell\upharpoonright_\mathsf{p} \in \Lambda$ and $\ell.\mathsf{G}_i^\ell\upharpoonright_\mathsf{q} \in \Lambda'$ for all i, $1 \leq i \leq n$, and all $\ell \in msg(\Lambda)$. We get $\mathsf{G} \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathsf{r} \to \mathsf{s} : \{\ell_i.\mathsf{G}_i^\ell \mid 1 \leq i \leq n\}$ using rule [ICOMM].
(ii). The proof is by induction on $depth(\mathsf{G}, \mathsf{p})$ and by cases on the reduction rules. The case of rule [ECOMM] is easy. For rule [ICOMM], by the induction hypothesis, $\mathsf{G}_i\upharpoonright_\mathsf{p} = \mathsf{q}!\Lambda_i$ and $\mathsf{G}_i\upharpoonright_\mathsf{q} = \mathsf{p}?\Lambda_i'$ and $\ell \in msg(\Lambda_i) = msg(\Lambda_i')$ for all i, $1 \leq i \leq n$. By definition of projection $\mathsf{G}_i\upharpoonright_\mathsf{p} = \mathsf{G}_1\upharpoonright_\mathsf{p}$ and $\mathsf{G}_i\upharpoonright_\mathsf{q} = \mathsf{G}_1\upharpoonright_\mathsf{q}$ for all i, $1 \leq i \leq n$. Again by definition of projection $\mathsf{G}\upharpoonright_\mathsf{p} = \mathsf{G}_1\upharpoonright_\mathsf{p}$ and $\mathsf{G}\upharpoonright_\mathsf{q} = \mathsf{G}_1\upharpoonright_\mathsf{q}$. □

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

Theorem 4.8 (Subject Reduction). *If* $\vdash \mathcal{M} : \mathsf{G}$ *and* $\mathcal{M} \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathcal{M}'$, *then* $\mathsf{G} \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathsf{G}'$ *and* $\vdash \mathcal{M}' : \mathsf{G}'$.

Proof. First, we can deduce some structural information on $\mathcal{M}$ and $\mathcal{M}'$. Indeed, since $\mathcal{M} \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathcal{M}'$, then $\mathsf{p} \triangleright \mathsf{q}!(\ell.P \uplus \Lambda) \in \mathcal{M}$, $\mathsf{q} \triangleright \mathsf{p}?(\ell.Q \uplus \Lambda') \in \mathcal{M}$ and $\mathsf{p} \triangleright P \in \mathcal{M}'$, $\mathsf{q} \triangleright Q \in \mathcal{M}'$. Moreover $\mathsf{r} \triangleright R \in \mathcal{M}$ iff $\mathsf{r} \triangleright R \in \mathcal{M}'$ for all $\mathsf{r} \notin \{\mathsf{p}, \mathsf{q}\}$. Thanks to typing, we can deduce similar information on G. Indeed, since $\vdash \mathcal{M} : \mathsf{G}$, we have that $\mathsf{q}!(\ell.P \uplus \Lambda) \leqslant \mathsf{G}\upharpoonright_\mathsf{p}$, and $\mathsf{p}?(\ell.Q \uplus \Lambda') \leqslant \mathsf{G}\upharpoonright_\mathsf{q}$, and for all $\mathsf{r} \triangleright R \in \mathcal{M}$ such that $\mathsf{r} \notin \{\mathsf{p}, \mathsf{q}\}$ we have $R \leqslant \mathsf{G}\upharpoonright_\mathsf{r}$ by Lemma 4.1. By definition of $\leqslant$, from $\mathsf{q}!(\ell.P \uplus \Lambda) \leqslant \mathsf{G}\upharpoonright_\mathsf{p}$ we get $\mathsf{G}\upharpoonright_\mathsf{p} = \mathsf{q}!(\ell.P_0 \uplus \Lambda_0)$ and $P \leqslant P_0$. Similarly from $\mathsf{p}?(\ell.Q \uplus \Lambda') \leqslant \mathsf{G}\upharpoonright_\mathsf{q}$ we get $\mathsf{G}\upharpoonright_\mathsf{q} = \mathsf{p}?(\ell.Q_0 \uplus \Lambda_0')$ and $Q \leqslant Q_0$. Lemma 4.7(i) implies $\mathsf{G} \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathsf{G}'$ and $\mathsf{G}'\upharpoonright_\mathsf{p} = P_0$ and $\mathsf{G}'\upharpoonright_\mathsf{q} = Q_0$. This proves the first part of the conclusion, we now move to the second one. We show $\mathsf{G}\upharpoonright_\mathsf{r} \leqslant \mathsf{G}'\upharpoonright_\mathsf{r}$ for each $\mathsf{r} \notin \{\mathsf{p}, \mathsf{q}\}$ and $\mathsf{r} \triangleright R \in \mathcal{M}$ by induction on $depth(\mathsf{G}, \mathsf{r})$ and by cases on the reduction rules. For rule [ECOMM] we get $\mathsf{G} = \mathsf{p} \to \mathsf{q} : (\ell.\mathsf{G}' \uplus \Gamma)$. By Definition 3.3 either $\mathsf{G}\upharpoonright_\mathsf{r} = \mathsf{G}'\upharpoonright_\mathsf{r}$ or $\mathsf{G}\upharpoonright_\mathsf{r} \leqslant \mathsf{G}'\upharpoonright_\mathsf{r}$. For rule [ICOMM] $\mathsf{G}_i\upharpoonright_\mathsf{r} \leqslant \mathsf{G}_i'\upharpoonright_\mathsf{r}$ for all i, $1 \leq i \leq n$, by the induction hypothesis. This implies $\mathsf{G}\upharpoonright_\mathsf{r} \leqslant \mathsf{G}'\upharpoonright_\mathsf{r}$. We conclude $\vdash \mathcal{M}' : \mathsf{G}'$. □

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

Theorem 4.9 (Session Fidelity). *Let* $\vdash \mathcal{M} : \mathsf{G}$.

(i) If $\mathcal{M} \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathcal{M}'$, *then* $\mathsf{G} \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathsf{G}'$ *and* $\vdash \mathcal{M}' : \mathsf{G}'$.

(ii) If $\mathsf{G} \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathsf{G}'$, *then* $\mathcal{M} \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathcal{M}'$ *and* $\vdash \mathcal{M}' : \mathsf{G}'$.

Proof. (i). It is the Subject Reduction Theorem.
(ii). By Lemma 4.7(ii), $\mathsf{G}\upharpoonright_{\mathsf{p}} = \mathsf{q}!\Lambda$ and $\mathsf{G}\upharpoonright_{\mathsf{q}} = \mathsf{p}?\Lambda'$ and $\ell \in msg(\Lambda) = msg(\Lambda')$. By Lemma 4.7(i) $\ell.\mathsf{G}'\upharpoonright_{\mathsf{p}} \in \Lambda$ and $\ell.\mathsf{G}'\upharpoonright_{\mathsf{q}} \in \Lambda'$. By Lemma 4.2 $\mathsf{p} \triangleright P \in \mathcal{M}$ and $\mathsf{q} \triangleright Q \in \mathcal{M}$ and $P \leqslant \mathsf{G}\upharpoonright_{\mathsf{p}}$ and $Q \leqslant \mathsf{G}\upharpoonright_{\mathsf{q}}$. By definition of $\leqslant$ we get

- $P = \mathsf{q}!(\ell.P' \uplus \Lambda_1)$ with $\{\ell\} \cup msg(\Lambda_1) = msg(\Lambda)$ and $P' \leqslant \mathsf{G}'\upharpoonright_{\mathsf{p}}$,
- $Q = \mathsf{p}?(\ell.Q' \uplus \Lambda_2)$ with $\{\ell\} \cup msg(\Lambda_2) \supseteq msg(\Lambda')$ and $Q' \leqslant \mathsf{G}'\upharpoonright_{\mathsf{q}}$.

Hence we have $\mathcal{M} \xrightarrow{\mathsf{p}\ell\mathsf{q}} \mathcal{M}'$ with $\mathsf{p} \triangleright P' \in \mathcal{M}'$, $\mathsf{q} \triangleright Q' \in \mathcal{M}'$ and $\mathsf{r} \triangleright R \in \mathcal{M}$ iff $\mathsf{r} \triangleright R \in \mathcal{M}'$ for all $\mathsf{r} \notin \{\mathsf{p}, \mathsf{q}\}$. We conclude $\vdash \mathcal{M}' : \mathsf{G}'$. □

We end this section by showing that the type system assures lock-freedom. By Subject Reduction it is enough to prove that in well-typed sessions no active participant waits forever. It follows from Session Fidelity, using Lemma 4.6.

Theorem 4.10 (Lock-freedom). *If* $\mathcal{M}$ *is well-typed, then* $\mathcal{M}$ *is lock-free.*

Proof. Let $\mathcal{M} \longrightarrow^* \mathcal{M}'' \equiv \mathsf{p} \triangleright P \mid \mathcal{M}'$ with $P \neq \mathbf{0}$. According to Definition 2.5(ii) (lock-freedom), we have to show that $\mathsf{p} \triangleright P \mid \mathcal{M}' \longrightarrow^* \mathcal{M}'' \xrightarrow{\lambda}$ where p occurs in λ. By Subject Reduction $\mathcal{M}''$ can be typed. Let G be a type for $\mathcal{M}''$. Since $\mathcal{M}'' \not\equiv \mathsf{p} \triangleright \mathbf{0}$, we can infer that $\mathsf{G} \neq \mathtt{End}$. We proceed now by induction on $w = depth(\mathsf{G}, \mathsf{p})$.
If $w = 0$ then either $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \Gamma$ or $\mathsf{G} = \mathsf{q} \to \mathsf{p} : \Gamma$ and $\mathsf{G} \xrightarrow{\lambda} \mathsf{G}'$ with p in λ by rule [ECOMM]. Then $\mathcal{M}'' \xrightarrow{\lambda} \mathcal{M}''$ by Theorem 4.9(ii).
If $w > 0$ then $\mathsf{G} = \mathsf{q} \to \mathsf{r} : \Gamma$ with $\mathsf{p} \notin \{\mathsf{q}, \mathsf{r}\}$ and $\mathsf{G} \xrightarrow{\mathsf{q}\ell\mathsf{r}} \mathsf{G}^\ell$ for all $\ell.\mathsf{G}^\ell \in \Gamma$ by rule [ECOMM]. By Lemma 4.6 $depth(\mathsf{G}, \mathsf{p}) > depth(\mathsf{G}^\ell, \mathsf{p})$ and induction applies. □

It is easy to check that $\vdash \mathcal{M} : \mathsf{G}$, where $\mathcal{M}$ and G are the multiparty session and the global type of Examples 2.4 and 3.2, respectively. By the above result, $\mathcal{M}$ of Example 2.4 is hence provably lock-free.

5. Composition of Multiparty-Sessions via Gateways

Two multiparty sessions can be *composed via gateways* when they possess two *compatible* participants, i.e. participants that offer communications which can be paired. Hence, the two participants can be transformed into forwarders, that we dub "gateways".

We start by discussing the relation of compatibility between processes by elaborating on Examples 2.4 and 3.2. Consider the participant h in these examples as an interface; the messages sent by h have to be considered as those actually provided by an external environment, and the received messages as messages expected by such an environment. In a sense, this means that, if we abstract from participants' name in the process $H = \mathsf{q}?\mathit{text}.H_1$ where $H_1 = \mathsf{q}!\{\mathit{ack}.H, \mathit{nack}.\mathsf{q}?\mathit{transf}.H_1, \mathit{stop}\}$, we get an interface towards an external system rather than a description of an internal component of our system.

In order to better grasp the notion of compatibility hinted at above, let $\textsc{an}(\_)$ be an operation abstracting from the participants' name inside processes. So, in our example we would get

$$\begin{array}{lcl} \textsc{an}(H) & = & \circ?\mathit{text}.\textsc{an}(H_1) \\ \textsc{an}(H_1) & = & \circ!\{\mathit{ack}.\textsc{an}(H), \mathit{nack}.\circ?\mathit{transf}.\textsc{an}(H_1), \mathit{stop}\} \end{array}$$

where $\circ$ stands for an abstracted participant name.
The following example gives another system that could work as the environment of the system in Example 2.4.

Example 5.1. Consider a system formed by participants k, r and s interacting according to the following protocol:

Participant k sends text messages to r and s in an alternating way, starting with r.

Participants r and s inform k that a text has been accepted or refused by sending back, respectively, either *ack* or *nack*.

- In the first case it is the other receiver's turn to receive the text: a message *go* is exchanged between r and s to signal this case;
- In the second case, the sender has to resend the text until it is accepted. Meanwhile the involved participant between r and s informs the other one that she needs to *wait*, since the previous message is being resent in a *transf*ormed form.

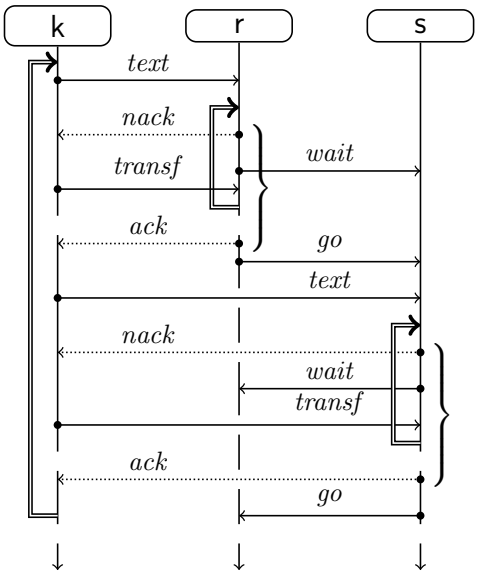


Figure 6: UML-like diagram for Example 5.1

In order to better understand the above protocol, we can describe it in UML-like form as done in Figure 6.

This protocol can be implemented by the multiparty session

$$\mathcal{M}' = \mathsf{r} \triangleright R \mid \mathsf{s} \triangleright S \mid \mathsf{k} \triangleright K_{\mathsf{r}}$$

where

$$\begin{array}{ll} R = \mathsf{k}?\mathit{text}.R_1 & R_1 = \mathsf{k}!\{\mathit{ack}.\mathsf{s}!\mathit{go}.R_2, \mathit{nack}.\mathsf{s}!\mathit{wait}.\mathsf{k}?\mathit{transf}.R_1\} \\ & R_2 = \mathsf{s}?\{\mathit{go}.R, \mathit{wait}.R_2\} \\ S = \mathsf{r}?\{\mathit{go}.\mathsf{k}?\mathit{text}.S_1, \mathit{wait}.S\} & S_1 = \mathsf{k}!\{\mathit{ack}.\mathsf{r}!\mathit{go}.S, \mathit{nack}.\mathsf{r}!\mathit{wait}.\mathsf{k}?\mathit{transf}.S_1\} \\ K_{\mathsf{r}} = \mathsf{r}!\mathit{text}.K'_{\mathsf{r}} & K'_{\mathsf{r}} = \mathsf{r}?\{\mathit{ack}.K_{\mathsf{s}}, \mathit{nack}.\mathsf{r}!\mathit{transf}.K'_{\mathsf{r}}\} \\ K_{\mathsf{s}} = \mathsf{s}!\mathit{text}.K'_{\mathsf{s}} & K'_{\mathsf{s}} = \mathsf{s}?\{\mathit{ack}.K_{\mathsf{r}}, \mathit{nack}.\mathsf{s}!\mathit{transf}.K'_{\mathsf{s}}\} \end{array}$$

◇

The "behaviour as interface" of participant k in the previous system corresponds to

$$\begin{array}{rcl} \text{AN}(K_{\mathsf{r}}) & = & \text{AN}(K_{\mathsf{s}}) = \circ!\mathit{text}.\text{AN}(K'_{\mathsf{r}}) \\ \text{AN}(K'_{\mathsf{r}}) & = & \text{AN}(K'_{\mathsf{s}}) = \circ?\{\mathit{ack}.\text{AN}(K_{\mathsf{r}}), \mathit{nack}.\circ!\mathit{transf}.\text{AN}(K'_{\mathsf{r}})\} \end{array}$$

Notice that the mapping AN equates K_{r} and K_{s}, i.e. $\text{AN}(K_{\mathsf{r}}) = \text{AN}(K_{\mathsf{s}})$.
The interactions "offered" and "requested" by $\text{AN}(H)$ and $\text{AN}(K_{\mathsf{r}})$ do not precisely match each other, that is $\overline{\text{AN}(H)} \neq \text{AN}(K_{\mathsf{r}})$ (where $\overline{(\cdot)}$ is the standard syntactic duality function replacing '!' by '?' and vice versa [30]). Nonetheless, it is easy to check that, even if the system $\mathsf{p} \triangleright P \mid \mathsf{q} \triangleright Q$ of Example 2.4 can safely deal with a message *stop* coming from its environment, no problem arises in case such a message never arrives.

In the following definition, instead of explicitly introducing the $\text{AN}(\_)$ function, we simply formalise the compatibility relation in such a way that two processes are compatible (as interfaces) whenever they offer dual communications to *arbitrary* participants, and, for each communication, the set of input labels is a subset of the set of output labels. This allows one gateway to send to the other each message received and to receive from the other each message she will send, see Definition 5.4.

Definition 5.2 (Compatible Processes). *The* interface compatibility relation $P \leftrightarrow Q$ *on processes (*compatibility *for short), is the largest symmetric relation coinductively defined by:*

$$[\text{COMP-}\mathbf{0}] \quad \mathbf{0} \leftrightarrow \mathbf{0} \qquad\qquad [\text{COMP-O/I}] \quad \frac{P_i \leftrightarrow Q_i \quad \forall 1 \leq i \leq n}{\mathsf{p}!(\{\ell_i.P_i \mid {\scriptstyle 1 \leq i \leq n}\} \uplus \Lambda) \leftrightarrow \mathsf{q}?\{\ell_i.Q_i \mid {\scriptstyle 1 \leq i \leq n}\}}$$

Notice that the relation $\_ \leftrightarrow \_$ is insensitive to the names of senders and receivers. Process compatibility is similar to, but simpler than, the subtyping relation defined in [28]. Therefore one can easily adapt the algorithm for subtyping in [28] so to check process compatibility.
For what concerns our example, it is straightforward to verify that $H \leftrightarrow K_{\mathsf{r}}$.

Useful properties of compatibility are stated in the following proposition, whose proof is simple.

Proposition 5.3 (Compatibility Properties). *(i) If* $P \leftrightarrow \mathsf{p}?(\Lambda \uplus \Lambda')$, *then* $P \leftrightarrow \mathsf{p}?\Lambda$.

(ii) If $P \leftrightarrow \mathsf{p}?\Lambda$ *and* $P \leftrightarrow \mathsf{p}?\Lambda'$ *with* $\Lambda \cap \Lambda' = \emptyset$, *then* $P \leftrightarrow \mathsf{p}?(\Lambda \uplus \Lambda')$.

(iii) If $\mathsf{p}!(\ell.P \uplus \Lambda) \leftrightarrow \mathsf{q}?\ell.Q$, *then* $P \leftrightarrow Q$.

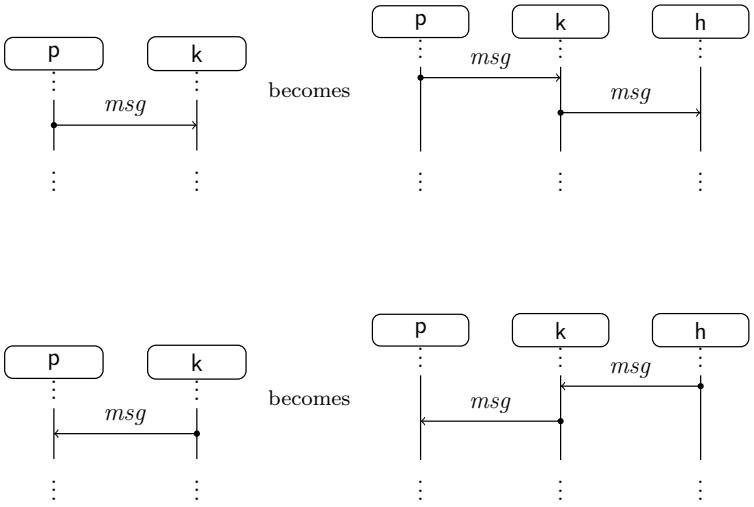


Figure 7: UML-like diagrams for Definition 5.4

As done in [1] for the setting of CFSMs [9], the presence of two compatible processes H and K in two multiparty sessions $\mathcal{M}$ and $\mathcal{M}'$ enables their composition by transforming H and K into a pair of gateways. This means that each message received by H is immediately sent to K, each message sent by H needs to be first received from K, and the same holds also with H and K swapped. Hence, we define below a function $\mathsf{gw}(P, \mathsf{h})$ which transforms an arbitrary process P (representing the behaviour of K above) not containing a fixed participant h into a gateway towards h, namely a process which:

1. sends to h each message received in P;
2. receives from h each message to be sent in P.

Considering the process of participant k chatting with participant p, this transformation can be represented by the UML-like diagrams of Figure 7.

Definition 5.4 (Gateway Process). *Let* $\mathsf{h} \notin \mathtt{ptp}(P)$. *We define the gateway process* $\mathsf{gw}(P, \mathsf{h})$ *coinductively as follows*

$$\begin{array}{lcl} \mathsf{gw}(\mathbf{0}, \mathsf{h}) & = & \mathbf{0} \\ \mathsf{gw}(\mathsf{p}?\{\ell_i.P_i \mid {\scriptstyle 1 \le i \le n}\}, \mathsf{h}) & = & \mathsf{p}?\{\ell_i.\mathsf{h}!\ell_i.\mathsf{gw}(P_i, \mathsf{h}) \mid {\scriptstyle 1 \le i \le n}\} \\ \mathsf{gw}(\mathsf{p}!\{\ell_i.P_i \mid {\scriptstyle 1 \le i \le n}\}, \mathsf{h}) & = & \mathsf{h}?\{\ell_i.\mathsf{p}!\ell_i.\mathsf{gw}(P_i, \mathsf{h}) \mid {\scriptstyle 1 \le i \le n}\} \end{array}$$

A first lemma assures that gateway processes are well-defined.

Lemma 5.5 (Gateway Processes are Well-Defined). *If* $\mathsf{h} \notin \mathtt{ptp}(P)$*, then* $\mathsf{gw}(P, \mathsf{h})$ *is defined and it is a process.*

Proof. The proof is by coinduction on P and by cases on its shape. The case $P = \mathbf{0}$ is trivial. If $P = \mathsf{p}?\{\ell_i.P_i \mid {}_{1\leq i\leq n}\}$ then

$$\mathsf{gw}(P, \mathsf{h}) = \mathsf{p}?\{\ell_i.\mathsf{h}!\ell_i.\mathsf{gw}(P_i, \mathsf{h}) \mid {}_{1\leq i\leq n}\}$$

For each i, $1 \leq i \leq n$, by coinduction $\mathsf{gw}(P_i, \mathsf{h})$ is defined and it is a process, since $\mathsf{h} \notin \mathtt{ptp}(P_i)$. Hence, by definition also $\mathsf{gw}(\mathsf{p}?\{\ell_i.P_i \mid {}_{1\leq i\leq n}\}, \mathsf{h})$ is a process. The proof for the case where P is an output process is similar. □

The gateway process construction is monotone with respect to the structural preorder. This property is key to get Theorem 6.10 (ensuring that composition via gateways preserves typability) and it essentially relies on the fact that larger processes offer the same output messages than smaller ones.

Lemma 5.6 (Monotonicity of Gateway Construction).
Let $\mathsf{h} \notin \mathtt{ptp}(P) \cup \mathtt{ptp}(Q)$*. If* $P \leqslant Q$*, then* $\mathsf{gw}(P, \mathsf{h}) \leqslant \mathsf{gw}(Q, \mathsf{h})$*.*

Proof. The proof is by coinduction on the derivation of $P \leqslant Q$. We only consider the case of input processes, the proof for output processes is similar and simpler, the one for $\mathbf{0}$ is trivial.
If $P = \mathsf{p}?\{\ell_i.P_i \mid {}_{1\leq i\leq n}\}$ and $Q = \mathsf{p}?\{\ell_i.Q_i \mid {}_{1\leq i\leq m}\}$ with $m \leq n$, then

$$\begin{aligned}\mathsf{gw}(P, \mathsf{h}) =& \mathsf{p}?\{\ell_i.\mathsf{h}!\ell_i.\mathsf{gw}(P_i, \mathsf{h}) \mid {}_{1\leq i\leq n}\} \text{ and}\\ \mathsf{gw}(Q, \mathsf{h}) =& \mathsf{p}?\{\ell_i.\mathsf{h}!\ell_i.\mathsf{gw}(Q_i, \mathsf{h}) \mid {}_{1\leq i\leq m}\}\end{aligned}$$

From $P \leqslant Q$ we get $P_i \leqslant Q_i$ for all i, $1 \leq i \leq m$. We get by coinduction $\mathsf{gw}(P_i, \mathsf{h}) \leqslant \mathsf{gw}(Q_i, \mathsf{h})$, which implies $\mathsf{h}!\ell_i.\mathsf{gw}(P_i, \mathsf{h}) \leqslant \mathsf{h}!\ell_i.\mathsf{gw}(Q_i, \mathsf{h})$ for all i, $1 \leq i \leq m$. Hence $\mathsf{gw}(P, \mathsf{h}) \leqslant \mathsf{gw}(Q, \mathsf{h})$, by Definition 3.9. □

The following relationship between compatibility and structural preorder of processes will be essential for proving one of our main results (Theorem 6.10).

Lemma 5.7 (Compatibility is Upward-Closed). *If* $P \leftrightarrow Q$*, then* $P \leqslant P'$ *and* $Q \leqslant Q'$ *imply* $P' \leftrightarrow Q'$*.*

Proof. By definition of process compatibility (Definition 5.2) and of process structural preorder (Definition 3.9)

$$\begin{array}{lcl} P = \mathsf{p}!\{\ell_i.P_i \mid 1{\leq}i{\leq}n\} & \leqslant & P' = \mathsf{p}!\{\ell_i.P'_i \mid 1{\leq}i{\leq}n\} \\ \updownarrow & & \\ Q = \mathsf{q}?\{\ell_i.Q_i \mid 1{\leq}i{\leq}m\} & \leqslant & Q' = \mathsf{q}?\{\ell_i.Q'_i \mid 1{\leq}i{\leq}m'\} \quad \text{with } m' \leq m \leq n \end{array}$$

From $P \leftrightarrow Q$ we get $P_i \leftrightarrow Q_i$ for all i, $1 \leq i \leq m$. From $P \leqslant P'$ we get $P_i \leqslant P'_i$ for all i, $1 \leq i \leq n$. From $Q \leqslant Q'$ we get $Q_i \leqslant Q'_i$ for all i, $1 \leq i \leq m'$. By coinduction we have $P'_i \leftrightarrow Q'_i$ for all i, $1 \leq i \leq m'$. We can then conclude $P' \leftrightarrow Q'$. □

Essentially the result above shows that compatibility is upward-closed with respect to the structural preorder. However, it is not downward-closed. For example $\mathsf{p}!\ell \leftrightarrow \mathsf{q}?\ell$ and $\mathsf{q}?\{\ell, \ell'\} \leqslant \mathsf{q}?\ell$, but $\mathsf{p}!\ell \leftrightarrow \mathsf{q}?\{\ell, \ell'\}$ is false.

The formal definition of composition of multiparty sessions via gateways is based on the notion of process compatibility (Definition 5.2) and on the transformation of a process into a gateway (Definition 5.4).

Definition 5.8 (Compatible Multiparty Sessions). *Two multiparty sessions $\mathcal{M}$, $\mathcal{M}'$ are* compatible via the participants h and k *if*
$\mathtt{pts}(\mathcal{M}) \cap \mathtt{pts}(\mathcal{M}') = \emptyset$ *and* $\mathsf{h} \triangleright H \in \mathcal{M}$ *and* $\mathsf{k} \triangleright K \in \mathcal{M}'$ *with* $H \leftrightarrow K$.
We write $(\mathcal{M}, \mathsf{h}) \leftrightarrow (\mathcal{M}', \mathsf{k})$ *when $\mathcal{M}$, $\mathcal{M}'$ are compatible via* h *and* k.

Definition 5.9 (Composition Via Gateways of Multiparty Sessions). *Let* $\mathcal{M} \equiv \mathcal{M}_1 \mid \mathsf{h} \triangleright H$, $\mathcal{M}' \equiv \mathcal{M}'_1 \mid \mathsf{k} \triangleright K$ *and* $(\mathcal{M}, \mathsf{h}) \leftrightarrow (\mathcal{M}', \mathsf{k})$. *We define* $\mathcal{M} \,{}^{\mathsf{h}}\!\!\ltimes\!\!\rtimes^{\mathsf{k}} \mathcal{M}'$, *the composition of $\mathcal{M}$ and $\mathcal{M}'$ via gateways, through* h *and* k, *by*

$$\mathcal{M} \,{}^{\mathsf{h}}\!\!\ltimes\!\!\rtimes^{\mathsf{k}} \mathcal{M}' = \mathcal{M}_1 \mid \mathcal{M}'_1 \mid \mathsf{h} \triangleright \mathsf{gw}(H, \mathsf{k}) \mid \mathsf{k} \triangleright \mathsf{gw}(K, \mathsf{h})$$

Example 5.10. Consider $\mathcal{M}$ of Example 2.4 and $\mathcal{M}'$ of Example 5.1. It is not difficult to check that

$$\mathcal{M} \,{}^{\mathsf{h}}\!\!\ltimes\!\!\rtimes^{\mathsf{k}} \mathcal{M}' \quad = \quad \mathsf{p} \triangleright P \mid \mathsf{q} \triangleright Q \mid \mathsf{r} \triangleright R \mid \mathsf{s} \triangleright S \mid \mathsf{h} \triangleright \hat{H} \mid \mathsf{k} \triangleright \hat{K}_{\mathsf{r}}$$

where

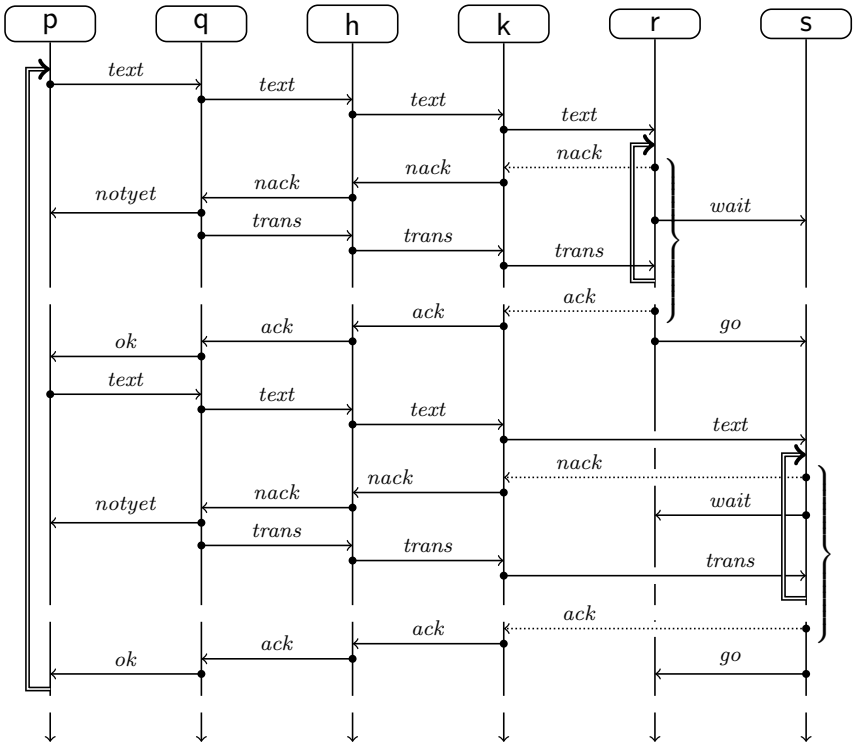


Figure 8: UML-like diagram of Example 5.10

$\hat{H} = \mathsf{gw}(H, \mathsf{k}) = \mathsf{q}?\mathit{text}.\mathsf{k}!\mathit{text}.\hat{H}_1 \qquad \hat{H}_1 = \mathsf{k}?\{\mathit{ack}.\mathsf{q}!\mathit{ack}.\hat{H},$
$\mathit{nack}.\mathsf{q}!\mathit{nack}.\mathsf{q}?\mathit{transf}.\mathsf{k}!\mathit{transf}.\hat{H}_1,$
$\mathit{stop}.\mathsf{q}!\mathit{stop}\}$
$\hat{K}_\mathsf{r} = \mathsf{gw}(K_\mathsf{r}, \mathsf{h}) = \mathsf{h}?\mathit{text}.\mathsf{r}!\mathit{text}.\hat{K}'_\mathsf{r} \qquad \hat{K}'_\mathsf{r} = \mathsf{r}?\{\mathit{ack}.\mathsf{h}!\mathit{ack}.\hat{K}_\mathsf{s},$
$\mathit{nack}.\mathsf{h}!\mathit{nack}.\mathsf{h}?\mathit{transf}.\mathsf{r}!\mathit{transf}.\hat{K}'_\mathsf{r}\}$
$\hat{K}_\mathsf{s} = \mathsf{gw}(K_\mathsf{s}, \mathsf{h}) = \mathsf{h}?\mathit{text}.\mathsf{s}!\mathit{text}.\hat{K}'_\mathsf{s} \qquad \hat{K}'_\mathsf{s} = \mathsf{s}?\{\mathit{ack}.\mathsf{h}!\mathit{ack}.\hat{K}_\mathsf{r},$
$\mathit{nack}.\mathsf{h}!\mathit{nack}.\mathsf{h}?\mathit{transf}.\mathsf{s}!\mathit{transf}.\hat{K}'_\mathsf{s}\}$

The overall behaviour of the composed system is described in the UML-like diagram in Figure 8. ◇

We have shown above how to compose two multiparty sessions, but we have not provided any guarantee on the behaviour of the resulting multiparty session. Composition via gateways preserves lock-freedom (cf. Section 6). To prove this, we define an operator taking the global types of two sessions and the name of a participant in each of them (the two participants need to be

compatible) and building a type for the composed multiparty session. The composed global type will directly mimic the diagram in Figure 8.

Notice that preservation of lock-freedom cannot be inferred from the results of [2], where lock-freedom was not taken into account.

6. Composition of Global Types via Gateways

The composition defined in the previous section can be shown to be lock-freedom preserving by means of Theorem 4.10. In fact, it is possible to lift the composition via gateways of multiparty sessions in Definition 5.9 to the level of global types. We start by defining the compatibility of global types via two participants, which mimics the compatibility of multiparty sessions via two participants

Definition 6.1 (Compatible Global Types). *Two global types* G, G' *are* compatible via the participants h and k *if* $\mathtt{ptg}(\mathsf{G}) \cap \mathtt{ptg}(\mathsf{G}') = \emptyset$ *and* $\mathsf{G}\upharpoonright_{\mathsf{h}} \leftrightarrow \mathsf{G}'\upharpoonright_{\mathsf{k}}$. *We write* $(\mathsf{G}, \mathsf{h}) \leftrightarrow (\mathsf{G}', \mathsf{k})$ *when* G, G' *are compatible via* h *and* k.

It is easy to verify that $(\mathsf{G}, \mathsf{h}) \leftrightarrow (\mathsf{G}', \mathsf{k})$ implies

$$\mathsf{h} \in \mathtt{ptg}(\mathsf{G}) \text{ if, and only if, } \mathsf{k} \in \mathtt{ptg}(\mathsf{G}').$$

The compatibility of typed sessions implies the compatibility of the corresponding global types.

Lemma 6.2 (Compatible Sessions have Compatible Types). *If* $(\mathcal{M}, \mathsf{h}) \leftrightarrow (\mathcal{M}', \mathsf{k})$ *and* $\vdash \mathcal{M} : \mathsf{G}$ *and* $\vdash \mathcal{M}' : \mathsf{G}'$, *then*

$$(\mathsf{G}, \mathsf{h}) \leftrightarrow (\mathsf{G}', \mathsf{k})$$

Proof. The typings $\vdash \mathcal{M} : \mathsf{G}$ and $\vdash \mathcal{M}' : \mathsf{G}'$ imply $\mathtt{ptg}(\mathsf{G}) \subseteq \mathtt{pts}(\mathcal{M})$ and $\mathtt{ptg}(\mathsf{G}') \subseteq \mathtt{pts}(\mathcal{M}')$ by rule [T-SESS]. Therefore we get that $\mathtt{pts}(\mathcal{M}) \cap \mathtt{pts}(\mathcal{M}') = \emptyset$ implies $\mathtt{ptg}(\mathsf{G}) \cap \mathtt{ptg}(\mathsf{G}') = \emptyset$. Let $\mathsf{h} \triangleright H \in \mathcal{M}$ and $\mathsf{k} \triangleright K \in \mathcal{M}'$. By the Inversion Lemma (Lemma 4.1) from $\vdash \mathcal{M} : \mathsf{G}$ we get $H \leqslant \mathsf{G}\upharpoonright_{\mathsf{h}}$ and from $\vdash \mathcal{M}' : \mathsf{G}'$ we get $K \leqslant \mathsf{G}'\upharpoonright_{\mathsf{k}}$. From $(\mathcal{M}, \mathsf{h}) \leftrightarrow (\mathcal{M}', \mathsf{k})$ we get $H \leftrightarrow K$ by Definition 5.8. Lemma 5.7 implies $\mathsf{G}\upharpoonright_{\mathsf{h}} \leftrightarrow \mathsf{G}'\upharpoonright_{\mathsf{k}}$. □

It is worth noticing that $(\mathsf{G}, \mathsf{h}) \leftrightarrow (\mathsf{G}', \mathsf{k})$ and $\vdash \mathcal{M} : \mathsf{G}$ and $\vdash \mathcal{M}' : \mathsf{G}'$ do not imply $(\mathcal{M}, \mathsf{h}) \leftrightarrow (\mathcal{M}', \mathsf{k})$, as shown in the following example.

Example 6.3. Take $\mathcal{M} = \mathsf{p} \triangleright \mathsf{h}?\ell \mid \mathsf{h} \triangleright \mathsf{p}!\ell$, $\mathcal{M}' = \mathsf{q} \triangleright \mathsf{k}!\ell \mid \mathsf{k} \triangleright \mathsf{q}?\{\ell, \ell'\}$, $\mathsf{G} = \mathsf{h} \to \mathsf{p} : \ell$, $\mathsf{G}' = \mathsf{q} \to \mathsf{k} : \ell$. Then $\mathsf{G} \upharpoonright_{\mathsf{p}} = \mathsf{h}?\ell$, $\mathsf{G} \upharpoonright_{\mathsf{h}} = \mathsf{p}!\ell$, $\mathsf{G}' \upharpoonright_{\mathsf{q}} = \mathsf{k}!\ell$, and $\mathsf{G} \upharpoonright_{\mathsf{k}} = \mathsf{q}?\ell$. We get $\mathsf{p}!\ell \leftrightarrow \mathsf{q}?\ell$, which implies $(\mathsf{G}, \mathsf{h}) \leftrightarrow (\mathsf{G}', \mathsf{k})$ according to Definition 6.1. Instead $\mathsf{p}!\ell \leftrightarrow \mathsf{q}?\{\ell, \ell'\}$ does not hold and therefore we do not have $(\mathcal{M}, \mathsf{h}) \leftrightarrow (\mathcal{M}', \mathsf{k})$ according to Definition 5.8. Intuitively, the typed sessions may have additional inputs due to structural preorder, and such additional inputs break compatibility at the level of sessions. $\diamond$

We are now ready to define the composition of global types via gateways. To avoid cumbersome parentheses, in the following we assume

$$\begin{gathered}
\star \to \star : \Gamma \mathbin{\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}} \star \to \star : \Gamma' \text{ reads } (\star \to \star : \Gamma) \mathbin{\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}} (\star \to \star : \Gamma') \\
\star \to \star : \Gamma \mathbin{\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}} \mathsf{G} \text{ reads } (\star \to \star : \Gamma) \mathbin{\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}} \mathsf{G} \\
\mathsf{G} \mathbin{\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}} \star \to \star : \Gamma \text{ reads } \mathsf{G} \mathbin{\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}} (\star \to \star : \Gamma) \\
\ell.\star \to \star : \ell.\star \to \star : \ell.\mathsf{G} \mathbin{\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}} \mathsf{G}' \text{ reads } \ell.\star \to \star : \ell.\star \to \star : \ell.(\mathsf{G} \mathbin{\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}} \mathsf{G}') \\
\ell.\mathsf{G} \mathbin{\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}} \mathsf{G}' \text{ reads } \ell.(\mathsf{G} \mathbin{\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}} \mathsf{G}')
\end{gathered}$$

where $\star$ stands for arbitrary, possibly different participant names.

Definition 6.4 (Global Type Composition via Gateways). *Let* $(\mathsf{G}, \mathsf{h}) \leftrightarrow (\mathsf{G}', \mathsf{k})$. *We define*

$$\mathsf{G} \mathbin{\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}} \mathsf{G}'$$

coinductively by the clauses of Figure 9, assuming $\{\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}\} \cap \{\mathsf{h}, \mathsf{k}\} = \emptyset$. *The clauses must be applied in the given order.*

---

(1) $\mathtt{End} \mathbin{\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}} \mathsf{G} = \mathsf{G}$

(2) $\mathsf{p} \to \mathsf{h} : \{\ell_i.\mathsf{G}_i \mid 1 \le i \le n\} \mathbin{\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}} \mathsf{k} \to \mathsf{s} : \{\ell_i.\mathsf{G}'_i \mid 1 \le i \le n\} \uplus \Gamma =$
$\mathsf{p} \to \mathsf{h} : \{\ell_i.\mathsf{h} \to \mathsf{k} : \ell_i.\mathsf{k} \to \mathsf{s} : \ell_i.\mathsf{G}_i \mathbin{\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}} \mathsf{G}'_i \mid 1 \le i \le n\}$

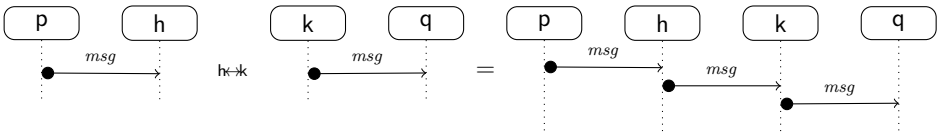
(3) $\mathsf{h} \to \mathsf{p} : \{\ell_i.\mathsf{G}_i \mid 1 \le i \le n\} \uplus \Gamma \mathbin{\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}} \mathsf{s} \to \mathsf{k} : \{\ell_i.\mathsf{G}'_i \mid 1 \le i \le n\} =$
$\mathsf{s} \to \mathsf{k} : \{\ell_i.\mathsf{k} \to \mathsf{h} : \ell_i.\mathsf{h} \to \mathsf{p} : \ell_i.\mathsf{G}_i \mathbin{\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}} \mathsf{G}'_i \mid 1 \le i \le n\}$

(4) $\mathsf{p} \to \mathsf{q} : \{\ell_i.\mathsf{G}_i \mid 1 \le i \le n\} \mathbin{\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}} \mathsf{G} = \mathsf{p} \to \mathsf{q} : \{\ell_i.\mathsf{G} \mathbin{\mathsf{k}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{h}} \mathsf{G}_i \mid 1 \le i \le n\}$

(5) $\mathsf{G} \mathbin{\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}} \mathsf{r} \to \mathsf{s} : \{\ell_i.\mathsf{G}_i \mid 1 \le i \le n\} = \mathsf{r} \to \mathsf{s} : \{\ell_i.\mathsf{G}_i \mathbin{\mathsf{k}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{h}} \mathsf{G} \mid 1 \le i \le n\}$

Figure 9: Definition of $\mathsf{h}\!\!\not\leftrightarrow\!\!\not\ast\mathsf{k}$ on global types

By Rule (2):



By Rule (3):

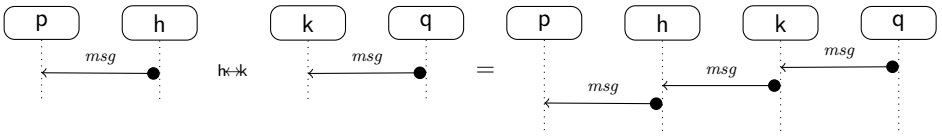


Figure 10: UML-like diagrams of Rules (2) and (3) in Definition 6.4

The core rules of our construction are rules (2) and (3), where the communication received by one of the two gateways is sent to the other one, which in turns outputs it. The effect of the application of such rules can be roughly interpreted in terms of diagrams as described in Figure 10.

Rules (4) and (5) allow a participant to take an interaction not involving the interface. In global types, the order of interactions between pairs of unrelated participants is irrelevant, since we would get the very same projections. In clauses (4) and (5), however, we swap roles h and k, as well as their corresponding global types in the "recursive call". We do that in order to spare the axiom $\mathsf{G} \setminus\!\!\!\leftrightarrow\!\!\!\!\!\!\ast\, \mathtt{End} = \mathsf{G}$. Moreover, this swapping avoids that in $\mathsf{G} \setminus\!\!\!\leftrightarrow\!\!\!\!\!\!\ast\, \mathsf{G}'$ the interactions following a communication via gateway all belong to G (or G') and that the communication is completed after the description of interactions all belonging to G' (or G). In this way, the parallel nature of the interactions in G and G' that are not affected by the communications via gateways is made visually more evident. Anyway the composition would be sound also without this swapping, just by adding the axiom $\mathsf{G} \setminus\!\!\!\leftrightarrow\!\!\!\!\!\!\ast\, \mathtt{End} = \mathsf{G}$.

Example 6.5. The protocol implemented by the multiparty session $\mathcal{M}'$ de-

fined in Example 5.1 can be represented by the following global type G_r:

$$\begin{aligned}
\mathsf{G}_\mathsf{r} &= \mathsf{k} \to \mathsf{r} : \mathit{text}.\mathsf{G}'_\mathsf{r} \\
\mathsf{G}'_\mathsf{r} &= \mathsf{r} \to \mathsf{k} : \{\mathit{ack}.\mathsf{r} \to \mathsf{s} : \mathit{go}.\mathsf{G}_\mathsf{s}, \\
&\qquad\qquad \mathit{nack}.\mathsf{r} \to \mathsf{s} : \mathit{wait}.\mathsf{k} \to \mathsf{r} : \mathit{transf}.\mathsf{G}'_\mathsf{r}\} \\
\mathsf{G}_\mathsf{s} &= \mathsf{k} \to \mathsf{s} : \mathit{text}.\mathsf{G}'_\mathsf{s} \\
\mathsf{G}'_\mathsf{s} &= \mathsf{s} \to \mathsf{k} : \{\mathit{ack}.\mathsf{s} \to \mathsf{r} : \mathit{go}.\mathsf{G}_\mathsf{r}, \\
&\qquad\qquad \mathit{nack}.\mathsf{s} \to \mathsf{r} : \mathit{wait}.\mathsf{k} \to \mathsf{s} : \mathit{transf}.\mathsf{G}'_\mathsf{s}\}
\end{aligned}$$

By composing, via h and k, the global type G of Example 3.2 and the above global type G_r we obtain a global type directly corresponding to the diagram in Figure 8. In particular, we get:

$$\mathsf{G} \,{}^{\mathsf{h}\nrightarrow\!\!\!\ast}\, \mathsf{G}_\mathsf{r} = \mathsf{p} \to \mathsf{q} : \mathit{text}.\mathsf{q} \to \mathsf{h} : \mathit{text}.\mathsf{h} \to \mathsf{k} : \mathit{text}.\mathsf{k} \to \mathsf{r} : \mathit{text}.\mathsf{G}'_\mathsf{r} \,{}^{\mathsf{k}\leftrightarrow\mathsf{h}}\, \mathsf{G}_1$$

$$\begin{aligned}
\mathsf{G}'_\mathsf{r} \,{}^{\mathsf{k}\leftrightarrow\mathsf{h}}\, \mathsf{G}_1 = \mathsf{r} \to \mathsf{k} : \{&\mathit{ack}.\mathsf{k} \to \mathsf{h} : \mathit{ack}.\mathsf{h} \to \mathsf{q} : \mathit{ack}.\mathsf{r} \to \mathsf{s} : \mathit{go}.\mathsf{q} \to \mathsf{p} : \mathit{ok}.\mathsf{G}_\mathsf{s} \,{}^{\mathsf{k}\leftrightarrow\mathsf{h}}\, \mathsf{G}, \\
&\mathit{nack}.\mathsf{k} \to \mathsf{h} : \mathit{nack}.\mathsf{h} \to \mathsf{q} : \mathit{nack}.\mathsf{r} \to \mathsf{s} : \mathit{wait}.\mathsf{q} \to \mathsf{p} : \mathit{notyet}. \\
&\mathsf{q} \to \mathsf{h} : \mathit{transf}.\mathsf{h} \to \mathsf{k} : \mathit{transf}.\mathsf{k} \to \mathsf{r} : \mathit{transf}.\mathsf{G}'_\mathsf{r} \,{}^{\mathsf{k}\leftrightarrow\mathsf{h}}\, \mathsf{G}_1\}
\end{aligned}$$

$$\mathsf{G}_\mathsf{s} \,{}^{\mathsf{k}\leftrightarrow\mathsf{h}}\, \mathsf{G} = \mathsf{p} \to \mathsf{q} : \mathit{text}.\mathsf{q} \to \mathsf{h} : \mathit{text}.\mathsf{h} \to \mathsf{k} : \mathit{text}.\mathsf{k} \to \mathsf{s} : \mathit{text}.\mathsf{G}_1 \,{}^{\mathsf{h}\nrightarrow\!\!\!\ast}\, \mathsf{G}'_\mathsf{s}$$

$$\begin{aligned}
\mathsf{G}_1 \,{}^{\mathsf{h}\nrightarrow\!\!\!\ast}\, \mathsf{G}'_\mathsf{s} = \mathsf{s} \to \mathsf{k} : \{&\mathit{ack}.\mathsf{k} \to \mathsf{h} : \mathit{ack}.\mathsf{h} \to \mathsf{q} : \mathit{ack}.\mathsf{q} \to \mathsf{p} : \mathit{ok}.\mathsf{s} \to \mathsf{r} : \mathit{go}.\mathsf{G} \,{}^{\mathsf{h}\nrightarrow\!\!\!\ast}\, \mathsf{G}_\mathsf{r}, \\
&\mathit{nack}.\mathsf{k} \to \mathsf{h} : \mathit{nack}.\mathsf{h} \to \mathsf{q} : \mathit{nack}.\mathsf{q} \to \mathsf{p} : \mathit{notyet}.\mathsf{s} \to \mathsf{r} : \mathit{wait}. \\
&\mathsf{q} \to \mathsf{h} : \mathit{transf}.\mathsf{h} \to \mathsf{k} : \mathit{transf}.\mathsf{k} \to \mathsf{s} : \mathit{transf}.\mathsf{G}_1 \,{}^{\mathsf{h}\nrightarrow\!\!\!\ast}\, \mathsf{G}'_\mathsf{s}\}
\end{aligned}$$

In $\mathsf{G} \,{}^{\mathsf{h}\nrightarrow\!\!\!\ast}\, \mathsf{G}_\mathsf{r}$ the text messages coming from p are delivered to q and, alternately, to r and s till they are accepted (*ack*). Participant p is informed when text messages are accepted (*ok*). During the cycle, q transforms a not yet accepted text into a more suitable form. The messages between q on one side and r and s on the other side are exchanged by passing through the coupled gateways h and k.

It is worth pointing out that in $\mathsf{G} \,{}^{\mathsf{h}\nrightarrow\!\!\!\ast}\, \mathsf{G}_\mathsf{r}$, the *stop* branch of G disappeared. In fact, since any message coming from h in G does now come from k (which is now the gateway forwarding the messages coming in turn from either r or s), the function ${}^{\mathsf{h}\nrightarrow\!\!\!\ast}$ takes care of the fact that only *ack* or *nack* can be received by (the gateway) h. This fact is reflected in the following Theorem 6.9, where it is shown that $\mathsf{gw}(\mathsf{G}\!\upharpoonright_\mathsf{h}, \mathsf{k})$ and $\mathsf{gw}(\mathsf{G}'\!\upharpoonright_\mathsf{k}, \mathsf{h})$ are less than or equal to the projections of $\mathsf{G} \,{}^{\mathsf{h}\nrightarrow\!\!\!\ast}\, \mathsf{G}'$ on h and k, respectively.

We could look at both h and p as interfaces: h representing a social-network system, which does not accept rude language, and p a social-network

client sending text messages and requiring to be informed about their delivery status. From this point of view, the global type G of Example 3.2 actually describes a "delivery-guaranteed" service for text messages, assuring messages to be eventually delivered by means of a text-transformation policy. $\diamond$

In order to show that composition via gateways is well-defined, it is handy to express the condition $\mathsf{G}\upharpoonright_{\mathsf{h}} \leftrightarrow \mathsf{G}'\upharpoonright_{\mathsf{k}}$ by means of a relation, dubbed agreement, between the pairs (G, h) and (G, k). The agreement relation is used in the proof of Lemma 6.8.

Definition 6.6 (Agreement Relation). *The agreement relation*

$$(\mathsf{G}, \mathsf{h}) \leftrightsquigarrow (\mathsf{G}, \mathsf{k})$$

is the largest symmetric relation coinductively defined by the rules:

$$\frac{}{(\mathtt{End}, \mathsf{h}) \leftrightsquigarrow (\mathsf{G}', \mathsf{k}) \ \textit{if}\ \mathsf{k} \notin \mathtt{ptg}(\mathsf{G}')}\ [\textsc{rel-End}]$$

$$\frac{\mathsf{h} \notin \{\mathsf{p}, \mathsf{q}\} \qquad (\mathsf{G}_i, \mathsf{h}) \leftrightsquigarrow (\mathsf{G}, \mathsf{k}) \quad \forall 1 \leq i \leq n}{(\mathsf{p} \to \mathsf{q} : \{\ell_i.\mathsf{G}_i \mid {\scriptstyle 1 \leq i \leq n}\}, \mathsf{h}) \leftrightsquigarrow (\mathsf{G}, \mathsf{k})}\ [\textsc{rel-go}]$$

$$\frac{(\mathsf{G}_i, \mathsf{h}) \leftrightsquigarrow (\mathsf{G}'_i, \mathsf{k}) \quad \forall 1 \leq i \leq n \leq m}{(\mathsf{p} \to \mathsf{h} : \{\ell_i.\mathsf{G}_i \mid {\scriptstyle 1 \leq i \leq n}\}, \mathsf{h}) \leftrightsquigarrow (\mathsf{k} \to \mathsf{q} : \{\ell_i.\mathsf{G}'_i \mid {\scriptstyle 1 \leq i \leq m}\}, \mathsf{k})}\ [\textsc{rel-comm}]$$

The following lemma connects the agreement relation between pairs of global type/participant with the compatibility of the projections of global types on the participants.

Lemma 6.7 (Agreement and Compatibility).

$$(\mathsf{G}, \mathsf{h}) \leftrightsquigarrow (\mathsf{G}', \mathsf{k}) \ \textit{iff}\ \mathsf{G}\upharpoonright_{\mathsf{h}} \leftrightarrow \mathsf{G}'\upharpoonright_{\mathsf{k}}\ .$$

Proof. From $(\mathsf{G}, \mathsf{h}) \leftrightsquigarrow (\mathsf{G}', \mathsf{k})$ we can show $\mathsf{G}\upharpoonright_{\mathsf{h}} \leftrightarrow \mathsf{G}'\upharpoonright_{\mathsf{k}}$ by coinduction and by cases on the rules of Definition 6.6.
If the last applied rule is [REL-GO] by coinduction $\mathsf{G}_i\upharpoonright_{\mathsf{h}} \leftrightarrow \mathsf{G}'\upharpoonright_{\mathsf{k}}$ for all i, $1 \leq i \leq n$. By Definition 3.3 we have two cases:

1. $\mathsf{G}\upharpoonright_{\mathsf{h}} = \mathsf{G}_1\upharpoonright_{\mathsf{h}}$ if $\mathsf{G}_i\upharpoonright_{\mathsf{h}} = \mathsf{G}_1\upharpoonright_{\mathsf{h}}$ for all i, $1 \leq i \leq n$;

2. $\mathsf{G}\upharpoonright_{\mathsf{h}} = \mathsf{s}?(\Lambda_1 \uplus \ldots \uplus \Lambda_n)$ if $\mathsf{G}_i\upharpoonright_{\mathsf{p}} = \mathsf{s}?\Lambda_i$ for all i, $1 \leq i \leq n$ and $msg(\Lambda_i) \cap msg(\Lambda_j) = \emptyset$ for all i,j, $1 \leq i \neq j \leq n$.

In case 1 we are done. In case 2 we conclude by repeatedly applying Proposition 5.3(ii).
If the last applied rule is [REL-COMM], then by coinduction $\mathsf{G}_i\upharpoonright_{\mathsf{h}} \leftrightarrow \mathsf{G}'_i\upharpoonright_{\mathsf{k}}$ for all i, $1 \leq i \leq n$. We get $\mathsf{G}\upharpoonright_{\mathsf{h}} = \mathsf{p}?\{\mathsf{G}_i\upharpoonright_{\mathsf{h}}|_{1\leq i\leq n}\}$ and $\mathsf{G}'\upharpoonright_{\mathsf{k}} = \mathsf{q}!\{\mathsf{G}'_i\upharpoonright_{\mathsf{k}}|_{1\leq i\leq m}\}$ by Definition 3.3. We can then derive $\mathsf{G}\upharpoonright_{\mathsf{h}} \leftrightarrow \mathsf{G}'\upharpoonright_{\mathsf{k}}$ using rule [COMP-O/I] of Definition 5.2.

Vice versa, if $\neg\big((\mathsf{G},\mathsf{h}) \leftrightsquigarrow (\mathsf{G}',\mathsf{k})\big)$ there exists a "failing derivation" using rules of Definition 6.6 with conclusion $(\mathsf{G},\mathsf{h}) \leftrightsquigarrow (\mathsf{G}',\mathsf{k})$ and with at least one finite branch ending with a judgment having one of the following forms (or the symmetric ones):

1. $(\mathtt{End},\mathsf{h}) \leftrightsquigarrow (\mathsf{Y},\mathsf{k})$ with $\mathsf{k} \in \mathtt{ptg}(\mathsf{Y})$;

2. $(\mathsf{p} \to \mathsf{h} : \Gamma, \mathsf{h}) \leftrightsquigarrow (\mathsf{s} \to \mathsf{k} : \Gamma', \mathsf{k})$;

3. $(\mathsf{h} \to \mathsf{p} : \Gamma, \mathsf{h}) \leftrightsquigarrow (\mathsf{k} \to \mathsf{s} : \Gamma', \mathsf{k})$;

4. $(\mathsf{p} \to \mathsf{h} : \Gamma, \mathsf{h}) \leftrightsquigarrow (\mathsf{k} \to \mathsf{s} : \Gamma', \mathsf{k})$ and there exists $\ell \in msg(\Gamma) \setminus msg(\Gamma')$.

In order to show $\neg(\mathsf{G}\upharpoonright_{\mathsf{h}} \leftrightarrow \mathsf{G}'\upharpoonright_{\mathsf{k}})$ it is enough to assume $\mathsf{G}\upharpoonright_{\mathsf{h}} \leftrightarrow \mathsf{G}'\upharpoonright_{\mathsf{k}}$ and to derive a contradiction by checking that the following statements hold:

a) If $\mathsf{k} \in \mathtt{ptg}(\mathsf{Y})$, then $\neg(\mathtt{End}\upharpoonright_{\mathsf{h}} \leftrightarrow \mathsf{Y}\upharpoonright_{\mathsf{k}})$;

b) If $\mathsf{Y} = \mathsf{p} \to \mathsf{h} : \Gamma$ and $\mathsf{Y}' = \mathsf{s} \to \mathsf{k} : \Gamma'$, then $\neg(\mathsf{Y}\upharpoonright_{\mathsf{h}} \leftrightarrow \mathsf{Y}'\upharpoonright_{\mathsf{k}})$;

c) If $\mathsf{Y} = \mathsf{h} \to \mathsf{p} : \Gamma$ and $\mathsf{Y}' = \mathsf{k} \to \mathsf{s} : \Gamma'$, then $\neg(\mathsf{Y}\upharpoonright_{\mathsf{h}} \leftrightarrow \mathsf{Y}'\upharpoonright_{\mathsf{k}})$;

d) If $\mathsf{Y} = \mathsf{p} \to \mathsf{h} : \Gamma$, $\mathsf{Y}' = \mathsf{k} \to \mathsf{s} : \Gamma'$ and there exists $\ell \in msg(\Gamma) \setminus msg(\Gamma')$, then $\neg(\mathsf{Y}\upharpoonright_{\mathsf{h}} \leftrightarrow \mathsf{Y}'\upharpoonright_{\mathsf{k}})$.

Statements *a*), *b*), *c*) and *d*) above descend easily from Definition 3.3 of projection and Definition 5.2 of compatibility between processes. □

Using the previous lemma we can give the following definition of compatible global types, which is alternative to Definition 6.1:

$$(\mathsf{G},\mathsf{h}) \leftrightarrow (\mathsf{G}',\mathsf{k}) \textit{ if } \mathtt{ptg}(\mathsf{G}) \cap \mathtt{ptg}(\mathsf{G}') = \emptyset \textit{ and } (\mathsf{G},\mathsf{h}) \leftrightsquigarrow (\mathsf{G}',\mathsf{k}).$$

We prove that the composition operation satisfies the constraints of the definition of global type (cf. Definition 3.1).

Lemma 6.8 (Composition via Gateways is Well-Defined). *Let* $(\mathsf{G},\mathsf{h}) \leftrightarrow (\mathsf{G}',\mathsf{k})$. *Then* $\mathsf{G} \mathbin{{}^{\mathsf{h}}\!\!\not\leftrightarrow^{*}} \mathsf{G}'$ *is defined and it is a global type.*

Proof. We start showing that the agreement relation is preserved by the recursive calls performed during the composition of global types.

Claim: Let $(\mathsf{G},\mathsf{h}) \leftrightsquigarrow (\mathsf{G}',\mathsf{k})$. Then for any call $\mathsf{Y} \mathbin{{}^{\mathsf{h}}\!\!\not\leftrightarrow^{*}} \mathsf{Y}'$ or $\mathsf{Y}' \mathbin{{}^{\mathsf{k}}\!\!\not\leftrightarrow^{*}} \mathsf{Y}$ in the tree of the recursive calls of $\mathsf{G} \mathbin{{}^{\mathsf{h}}\!\!\not\leftrightarrow^{*}} \mathsf{G}'$ we get $(\mathsf{Y},\mathsf{h}) \leftrightsquigarrow (\mathsf{Y}',\mathsf{k})$.

The proof of the Claim is by coinduction on the rules of Definition 6.6 and by cases on the rules of Definition 6.4.
Rule (2): $\mathsf{Y} \mathbin{{}^{\mathsf{h}}\!\!\not\leftrightarrow^{*}} \mathsf{Y}' = \mathsf{p} \to \mathsf{h} : \{\ell_i.\mathsf{h} \to \mathsf{k} : \ell_i.\mathsf{k} \to \mathsf{s} : \ell_i.\mathsf{Y}_i \mathbin{{}^{\mathsf{h}}\!\!\not\leftrightarrow^{*}} \mathsf{Y}'_i \mid {}_{1\leq i\leq n}\}$ where $\mathsf{Y} = \mathsf{p} \to \mathsf{h} : \{\ell_i.\mathsf{Y}_i \mid {}_{1\leq i\leq n}\}$ and $\mathsf{Y}' = \mathsf{k} \to \mathsf{s} : \{\ell_i.\mathsf{Y}'_i \mid {}_{1\leq i\leq n}\} \uplus \Gamma$. The relation $(\mathsf{Y},\mathsf{h}) \leftrightsquigarrow (\mathsf{Y}',\mathsf{k})$ is the conclusion of rule [REL-COMM] (cf. Definition 6.6) with premises $(\mathsf{Y}_i,\mathsf{h}) \leftrightsquigarrow (\mathsf{Y}'_i,\mathsf{k})$ for all i, $1 \leq i \leq n$.
The proof for rule (3) is similar to the proof for rule (2).
Rule (4): $\mathsf{p} \to \mathsf{q} : \{\ell_i.\mathsf{Y}_i \mid {}_{1\leq i\leq n}\} \mathbin{{}^{\mathsf{h}}\!\!\not\leftrightarrow^{*}} \mathsf{Y}' = \mathsf{p} \to \mathsf{q} : \{\ell_i.\mathsf{Y}' \mathbin{{}^{\mathsf{k}}\!\!\not\leftrightarrow^{\mathsf{h}}} \mathsf{Y}_i \mid {}_{1\leq i\leq n}\}$. Let $\mathsf{Y} = \mathsf{p} \to \mathsf{q} : \{\ell_i.\mathsf{Y}_i \mid {}_{1\leq i\leq n}\}$. The relation $(\mathsf{Y},\mathsf{h}) \leftrightsquigarrow (\mathsf{Y}',\mathsf{k})$ is the conclusion of rule [REL-GO] with premises $(\mathsf{Y}_i,\mathsf{h}) \leftrightsquigarrow (\mathsf{Y}',\mathsf{k})$ for all i, $1 \leq i \leq n$.
The proof for rule (5) is similar to the proof for rule (4). This ends the proof of the Claim.

We now show the lemma. From $(\mathsf{G},\mathsf{h}) \leftrightarrow (\mathsf{G}',\mathsf{k})$ we have $(\mathsf{G},\mathsf{h}) \leftrightsquigarrow (\mathsf{G}',\mathsf{k})$. By the Claim for all recursive calls $\mathsf{Y} \mathbin{{}^{\mathsf{h}}\!\!\not\leftrightarrow^{*}} \mathsf{Y}'$ in $\mathsf{G} \mathbin{{}^{\mathsf{h}}\!\!\not\leftrightarrow^{*}} \mathsf{G}'$ we always get global types Y,Y' such that $(\mathsf{Y},\mathsf{h}) \leftrightsquigarrow (\mathsf{Y}',\mathsf{k})$. We show that $(\mathsf{Y},\mathsf{h}) \leftrightsquigarrow (\mathsf{Y}',\mathsf{k})$ assures the applicability of one rule in Definition 6.4. The proof is by cases on the rules of Definition 6.6.
Rule [REL-End]: we can apply rule (1).
Rule [REL-GO]: we can apply rule (4) or (5).
Rule [REL-COMM]: we can apply rule (2) or (3).
Notice that the applicability of the rules in alternative is due to the symmetry of the relation $(\mathsf{Y},\mathsf{h}) \leftrightsquigarrow (\mathsf{Y}',\mathsf{k})$.

The regularity of the obtained term follows by observing that the regularity of G and G' implies that the tree of the recursive calls has no infinite path with pairwise distinct calls. □

The following theorem gives the key result concerning projections of types obtained by composing via gateways the global types G and G' through participants h and k. It says that:

- the gateway processes are equal to or smaller than the corresponding projections of $\mathsf{G} \between \mathsf{G}'$ on h and k;
- the projections of G and G' on all participants different from h and k are equal to or smaller than the corresponding projections of $\mathsf{G} \between \mathsf{G}'$.

Theorem 6.9 (Projection of Composition via Gateways).
If $(\mathsf{G}, \mathsf{h}) \leftrightarrow (\mathsf{G}', \mathsf{k})$*, then* $\mathsf{G} \between \mathsf{G}'$ *is well-formed. Moreover*

(i) $\mathsf{gw}(\mathsf{G}\upharpoonright_{\mathsf{h}}, \mathsf{k}) \leqslant (\mathsf{G} \between \mathsf{G}')\upharpoonright_{\mathsf{h}}$ *and* $\mathsf{gw}(\mathsf{G}'\upharpoonright_{\mathsf{k}}, \mathsf{h}) \leqslant (\mathsf{G} \between \mathsf{G}')\upharpoonright_{\mathsf{k}}$*;*

(ii) $\mathsf{G}\upharpoonright_{\mathsf{p}} \leqslant (\mathsf{G} \between \mathsf{G}')\upharpoonright_{\mathsf{p}}$ *and* $\mathsf{G}'\upharpoonright_{\mathsf{q}} \leqslant (\mathsf{G} \between \mathsf{G}')\upharpoonright_{\mathsf{q}}$ *,*
for any $\mathsf{p} \in \mathtt{ptg}(\mathsf{G})$ *and* $\mathsf{q} \in \mathtt{ptg}(\mathsf{G}')$ *such that* $\mathsf{p} \neq \mathsf{h}$ *and* $\mathsf{q} \neq \mathsf{k}$*.*

Proof. By Lemma 6.8 $\mathsf{G} \between \mathsf{G}'$ is a global type. We recall that by Definition 3.7 a global type is well-formed iff

- each participant has finite depth for all subterms of the global type;
- the global type is projectable on each participant.

It is easy to verify that if

$$w = max\{depth(\mathsf{G}_0, \mathsf{p}) \mid \mathsf{G}_0 \text{ is a subterm of } \mathsf{G} \text{ and } \mathsf{p} \in \mathtt{ptg}(\mathsf{G})\} \text{ and}$$
$$w' = max\{depth(\mathsf{G}'_0, \mathsf{p}) \mid \mathsf{G}'_0 \text{ is a subterm of } \mathsf{G}' \text{ and } \mathsf{p} \in \mathtt{ptg}(\mathsf{G}')\}$$

then $depth(\hat{\mathsf{G}}, \mathsf{p}) \leq 2(w + w')$ for all $\hat{\mathsf{G}}$ subterm of $\mathsf{G} \between \mathsf{G}'$ and all $\mathsf{p} \in \mathtt{ptg}(\mathsf{G}) \cup \mathtt{ptg}(\mathsf{G}')$.

We show the projectability of the composition by proving (i) and (ii).
(i). We consider only the case $\mathsf{gw}(\mathsf{G}\upharpoonright_{\mathsf{h}}, \mathsf{k}) \leqslant (\mathsf{G} \between \mathsf{G}')\upharpoonright_{\mathsf{h}}$ as the proof of $\mathsf{gw}(\mathsf{G}'\upharpoonright_{\mathsf{k}}, \mathsf{h}) \leqslant (\mathsf{G} \between \mathsf{G}')\upharpoonright_{\mathsf{k}}$ is specular. We prove $\mathsf{gw}(\mathsf{Y}\upharpoonright_{\mathsf{h}}, \mathsf{k}) \leqslant (\mathsf{Y} \between \mathsf{Y}')\upharpoonright_{\mathsf{h}}$ for any recursive call $\mathsf{Y} \between \mathsf{Y}'$ in $\mathsf{G} \between \mathsf{G}'$ by coinduction and by cases on the last applied rule.
Rules (1), (4) and (5) do not modify the communications of participant h, so coinduction easily applies.
Rule (2): $\mathsf{Y} \between \mathsf{Y}' = \mathsf{p} \to \mathsf{h} : \{\ell_i.\mathsf{h} \to \mathsf{k} : \ell_i.\mathsf{k} \to \mathsf{s} : \ell_i.\mathsf{Y}_i \between \mathsf{Y}'_i \mid {}_{1 \leq i \leq n}\}$ where

$\mathsf{Y} = \mathsf{p} \to \mathsf{h} : \{\ell_i.\mathsf{Y}_i \mid {\scriptstyle 1 \le i \le n}\}$ and $\mathsf{Y}' = \mathsf{k} \to \mathsf{s} : \{\ell_i.\mathsf{Y}'_i \mid {\scriptstyle 1 \le i \le n}\} \uplus \Gamma$.
Then $\mathsf{Y}\upharpoonright_{\mathsf{h}} = \mathsf{p}?\{\ell_i.\mathsf{Y}_i\upharpoonright_{\mathsf{h}} \mid {\scriptstyle 1 \le i \le n}\}$ by Definition 3.3.

$$\begin{array}{rcll}
\mathsf{gw}(\mathsf{Y}\upharpoonright_{\mathsf{h}}, \mathsf{k}) & = & \mathsf{p}?\{\ell_i.\mathsf{k}!\ell_i.\mathsf{gw}(\mathsf{Y}_i\upharpoonright_{\mathsf{h}}, \mathsf{k}) \mid {\scriptstyle 1 \le i \le n}\} & \text{by Definition 5.4} \\
& \leqslant & \mathsf{p}?\{\ell_i.\mathsf{k}!\ell_i.(\mathsf{Y}_i \mathbin{\mathsf{h}\!\!\not\!\!\ast} \mathsf{Y}'_i)\upharpoonright_{\mathsf{h}} \mid {\scriptstyle 1 \le i \le n}\} & \text{by rules [SUB-IN] and [SUB-OUT]} \\
& & & \text{of Definition 3.9 since} \\
& & & \text{by coinduction} \\
& & & \mathsf{gw}(\mathsf{Y}_i\upharpoonright_{\mathsf{h}}, \mathsf{k}) \leqslant (\mathsf{Y}_i \mathbin{\mathsf{h}\!\!\not\!\!\ast} \mathsf{Y}'_i)\upharpoonright_{\mathsf{h}} \\
& & & \text{for all } i,\ 1 \le i \le n \\
& = & (\mathsf{Y} \mathbin{\mathsf{h}\!\!\not\!\!\ast} \mathsf{Y}')\upharpoonright_{\mathsf{h}} & \text{by Definition 3.3.}
\end{array}$$

Rule (3): $\mathsf{Y} \mathbin{\mathsf{h}\!\!\not\!\!\ast} \mathsf{Y}' = \mathsf{s} \to \mathsf{k} : \{\ell_i.\mathsf{k} \to \mathsf{h} : \ell_i.\mathsf{h} \to \mathsf{p} : \ell_i.\mathsf{Y}_i \mathbin{\mathsf{h}\!\!\not\!\!\ast} \mathsf{Y}'_i \mid {\scriptstyle 1 \le i \le n}\}$, where $\mathsf{Y} = \mathsf{h} \to \mathsf{p} : \{\ell_i.\mathsf{Y}_i \mid {\scriptstyle 1 \le i \le m}\}$ and $\mathsf{Y}' = \mathsf{s} \to \mathsf{k} : \{\ell_i.\mathsf{Y}'_i \mid {\scriptstyle 1 \le i \le n}\}$ with $m \ge n$.
Then $\mathsf{Y}\upharpoonright_{\mathsf{h}} = \mathsf{p}!\{\ell_i.\mathsf{Y}_i\upharpoonright_{\mathsf{h}} \mid {\scriptstyle 1 \le i \le m}\}$ by Definition 3.3.

$$\begin{array}{rcll}
\mathsf{gw}(\mathsf{Y}\upharpoonright_{\mathsf{h}}, \mathsf{k}) & = & \mathsf{k}?\{\ell_i.\mathsf{p}!\ell_i.\mathsf{gw}(\mathsf{Y}_i\upharpoonright_{\mathsf{h}}, \mathsf{k}) \mid {\scriptstyle 1 \le i \le m}\} & \text{by Definition 5.4} \\
& \leqslant & \mathsf{k}?\{\ell_i.\mathsf{p}!\ell_i.\mathsf{gw}(\mathsf{Y}_i\upharpoonright_{\mathsf{h}}, \mathsf{k}) \mid {\scriptstyle 1 \le i \le n}\} & \text{by rule [SUB-IN]} \\
& \leqslant & \mathsf{k}?\{\ell_i.\mathsf{p}!\ell_i.(\mathsf{Y}_i \mathbin{\mathsf{h}\!\!\not\!\!\ast} \mathsf{Y}'_i)\upharpoonright_{\mathsf{h}} \mid {\scriptstyle 1 \le i \le n}\} & \text{by rules [SUB-IN] and [SUB-OUT]} \\
& & & \text{of Definition 3.9 since} \\
& & & \text{by coinduction} \\
& & & \mathsf{gw}(\mathsf{Y}_i\upharpoonright_{\mathsf{h}}, \mathsf{k}) \leqslant (\mathsf{Y}_i \mathbin{\mathsf{h}\!\!\not\!\!\ast} \mathsf{Y}'_i)\upharpoonright_{\mathsf{h}} \\
& & & \text{for all } i,\ 1 \le i \le n \\
& = & (\mathsf{Y} \mathbin{\mathsf{h}\!\!\not\!\!\ast} \mathsf{Y}')\upharpoonright_{\mathsf{h}} & \text{by Definition 3.3.}
\end{array}$$

(ii). We only show $\mathsf{G}\upharpoonright_{\mathsf{p}} \leqslant (\mathsf{G} \mathbin{\mathsf{h}\!\!\not\!\!\ast} \mathsf{G}')\upharpoonright_{\mathsf{p}}$ for $\mathsf{p} \in \mathsf{ptg}(\mathsf{G})$ and $\mathsf{p} \neq \mathsf{h}$. The proof of $\mathsf{G}\upharpoonright_{\mathsf{q}} \leqslant (\mathsf{G} \mathbin{\mathsf{h}\!\!\not\!\!\ast} \mathsf{G}')\upharpoonright_{\mathsf{q}}$ for $\mathsf{q} \in \mathsf{ptg}(\mathsf{G}')$ and $\mathsf{q} \neq \mathsf{k}$ is specular. Consider the recursive calls $\mathsf{Y} \mathbin{\mathsf{h}\!\!\not\!\!\ast} \mathsf{Y}'$ in $\mathsf{G} \mathbin{\mathsf{h}\!\!\not\!\!\ast} \mathsf{G}'$. We prove $\mathsf{Y}\upharpoonright_{\mathsf{p}} \leqslant (\mathsf{Y} \mathbin{\mathsf{h}\!\!\not\!\!\ast} \mathsf{Y}')\upharpoonright_{\mathsf{p}}$ by coinduction on Y, Y' and by cases on the last applied rule. The only rule which modifies the communications of p is rule (3):
$\mathsf{Y} \mathbin{\mathsf{h}\!\!\not\!\!\ast} \mathsf{Y}' = \mathsf{s} \to \mathsf{k} : \{\ell_i.\mathsf{k} \to \mathsf{h} : \ell_i.\mathsf{h} \to \mathsf{p} : \ell_i.\mathsf{Y}_i \mathbin{\mathsf{h}\!\!\not\!\!\ast} \mathsf{Y}'_i \mid {\scriptstyle 1 \le i \le n}\}$, where $\mathsf{Y} = \mathsf{h} \to \mathsf{p} : \{\ell_i.\mathsf{Y}_i \mid {\scriptstyle 1 \le i \le m}\}$ and $\mathsf{Y}' = \mathsf{s} \to \mathsf{k} : \{\ell_i.\mathsf{Y}'_i \mid {\scriptstyle 1 \le i \le n}\}$ with $m \ge n$.

$$\begin{array}{rcll}
\mathsf{Y}\upharpoonright_{\mathsf{p}} & = & \mathsf{h}?\{\ell_i.\mathsf{Y}_i\upharpoonright_{\mathsf{p}} \mid {\scriptstyle 1 \le i \le m}\} & \text{by Definition 3.3} \\
& \leqslant & \mathsf{h}?\{\ell_i.\mathsf{Y}_i\upharpoonright_{\mathsf{p}} \mid {\scriptstyle 1 \le i \le n}\} & \text{by rule [SUB-IN]} \\
& \leqslant & \mathsf{h}?\{\ell_i.(\mathsf{Y}_i \mathbin{\mathsf{h}\!\!\not\!\!\ast} \mathsf{Y}'_i)\upharpoonright_{\mathsf{p}} \mid {\scriptstyle 1 \le i \le n}\} & \text{by rule [SUB-IN] since by coinduction} \\
& & & \mathsf{Y}_i\upharpoonright_{\mathsf{p}} \leqslant (\mathsf{Y}_i \mathbin{\mathsf{h}\!\!\not\!\!\ast} \mathsf{Y}'_i)\upharpoonright_{\mathsf{p}} \text{ for all } i,\ 1 \le i \le n \\
& = & (\mathsf{Y} \mathbin{\mathsf{h}\!\!\not\!\!\ast} \mathsf{Y}')\upharpoonright_{\mathsf{p}} & \text{by Definition 3.3.}
\end{array}$$

□

We now show that by composing via gateways two well-typed sessions which are compatible, we get a session which is also well-typed. This is relevant, since well-typed sessions enjoy lock-freedom (Theorem 4.10).

Theorem 6.10 (Typing Gateway Composition). *If* $(\mathcal{M}, \mathsf{h}) \leftrightarrow (\mathcal{M}', \mathsf{k})$ *and* $\vdash \mathcal{M} : \mathsf{G}$ *and* $\vdash \mathcal{M}' : \mathsf{G}'$*, then* $\vdash \mathcal{M} \mathbin{{}^{\mathsf{h}}\!\!\twoheadrightarrow\!\!\!\!\leftarrow^{\mathsf{k}}} \mathcal{M}' : \mathsf{G} \mathbin{{}^{\mathsf{h}}\!\!\twoheadrightarrow\!\!\!\!\leftarrow^{\mathsf{k}}} \mathsf{G}'$.

Proof. Let $\mathcal{M} \equiv \mathcal{M}_1 \mid \mathsf{h} \triangleright H$ and $\mathcal{M}' \equiv \mathcal{M}'_1 \mid \mathsf{k} \triangleright K$. By construction

$$\mathcal{M} \mathbin{{}^{\mathsf{h}}\!\!\twoheadrightarrow\!\!\!\!\leftarrow^{\mathsf{k}}} \mathcal{M}' = \mathcal{M}_1 \mid \mathcal{M}'_1 \mid \mathsf{h} \triangleright \mathsf{gw}(H, \mathsf{k}) \mid \mathsf{k} \triangleright \mathsf{gw}(K, \mathsf{h})$$

From $\vdash \mathcal{M} : \mathsf{G}$ we get $H \leqslant \mathsf{G}\upharpoonright_{\mathsf{h}}$ and from $\vdash \mathcal{M}' : \mathsf{G}'$ we get $K \leqslant \mathsf{G}'\upharpoonright_{\mathsf{k}}$, both thanks to the Inversion Lemma (Lemma 4.1). Lemma 5.6 gives $\mathsf{gw}(H, \mathsf{k}) \leqslant \mathsf{gw}(\mathsf{G}\upharpoonright_{\mathsf{h}}, \mathsf{k})$ and $\mathsf{gw}(K, \mathsf{h}) \leqslant \mathsf{gw}(\mathsf{G}'\upharpoonright_{\mathsf{k}}, \mathsf{h})$. From $(\mathcal{M}, \mathsf{h}) \leftrightarrow (\mathcal{M}', \mathsf{k})$ by Lemma 6.2 we get $(\mathsf{G}, \mathsf{h}) \leftrightarrow (\mathsf{G}', \mathsf{k})$. We conclude $\vdash \mathcal{M} \mathbin{{}^{\mathsf{h}}\!\!\twoheadrightarrow\!\!\!\!\leftarrow^{\mathsf{k}}} \mathcal{M}' : \mathsf{G} \mathbin{{}^{\mathsf{h}}\!\!\twoheadrightarrow\!\!\!\!\leftarrow^{\mathsf{k}}} \mathsf{G}'$ using the relations on the projections of $\mathsf{G} \mathbin{{}^{\mathsf{h}}\!\!\twoheadrightarrow\!\!\!\!\leftarrow^{\mathsf{k}}} \mathsf{G}'$ given in Theorem 6.9. □

As a general application of the previous results, let us suppose to have two systems that correspond to multiparty sessions that are compatible via some participants (according to Definition 5.8) and that are well-typed (according to Definition 3.10). At this point, we can "deploy" the composed system (following Definition 5.9) without any further verification step, since Theorem 6.10 ensures that under such conditions we have a well-typed and hence lock-free composed system. Besides, we are able to provide the documentation (the global type) of the resulting system.

7. Compatibility of Global Types is Necessary

We have shown that the composition of typeable multiparty sessions, via the transformation of two participants into gateways, does preserve lock-freedom in case the two participants are compatible. One could wonder whether session compatibility, besides being a sufficient condition for ensuring such a preservation property, is also necessary. Example 6.3 shows that this is not the case.

Example 7.1 (Session Compatibility is not Necessary). Let us take $\mathcal{M} = \mathsf{p} \triangleright \mathsf{h}?\ell \mid \mathsf{h} \triangleright \mathsf{p}!\ell$ and $\mathcal{M}' = \mathsf{q} \triangleright \mathsf{k}!\ell \mid \mathsf{k} \triangleright \mathsf{q}?\{\ell, \ell'\}$ from Example 6.3. Then $(\mathcal{M}, \mathsf{h}) \leftrightarrow (\mathcal{M}', \mathsf{k})$ does not hold, and therefore $\mathcal{M} \mathbin{{}^{\mathsf{h}}\!\!\twoheadrightarrow\!\!\!\!\leftarrow^{\mathsf{k}}} \mathcal{M}'$ is undefined. Nonetheless, applying the gateway construction (cf. Definition 5.4), we can build

$$\mathsf{p} \triangleright \mathsf{h}?\ell \mid \mathsf{h} \triangleright \mathsf{k}?\ell.\mathsf{p}!\ell \mid \mathsf{q} \triangleright \mathsf{k}!\ell \mid \mathsf{k} \triangleright \mathsf{q}?\{\ell.\mathsf{h}!\ell, \ell'.\mathsf{h}!\ell'\}$$

which is lock-free, since its only reduction terminates with $\mathsf{p} \triangleright \mathbf{0}$. The reason is that even if potentially k could try to send ℓ' to p (via h), creating a deadlock,

in this session such a possibility is never taken, since q will never send ℓ' to k. Notice that in the same example $(\mathsf{G},\mathsf{h}) \leftrightarrow (\mathsf{G}',\mathsf{k})$ instead holds (recall that in Example 6.3 $\mathsf{G} = \mathsf{h} \to \mathsf{p} : \ell$ and $\mathsf{G}' = \mathsf{q} \to \mathsf{k} : \ell$). $\diamond$

We will now show that compatibility of global types is instead necessary. Let $\neg\big((\mathsf{G},\mathsf{h}) \leftrightarrow (\mathsf{G}',\mathsf{k})\big)$ and $\vdash \mathcal{M} : \mathsf{G}$ and $\vdash \mathcal{M}' : \mathsf{G}'$. Lemma 6.2 gives $\neg\big((\mathcal{M},\mathsf{h}) \leftrightarrow (\mathcal{M}',\mathsf{k})\big)$. This implies that $\mathcal{M} \mathbin{\text{ɦ↔ɦ}} \mathcal{M}'$ is undefined. Anyway, if $\mathcal{M} \equiv \mathcal{M}_1 \mid \mathsf{h} \triangleright H$ and $\mathcal{M}' \equiv \mathcal{M}'_1 \mid \mathsf{k} \triangleright K$ we can build

$$\mathcal{M}_1 \mid \mathcal{M}'_1 \mid \mathsf{h} \triangleright \mathsf{gw}(H,\mathsf{k}) \mid \mathsf{k} \triangleright \mathsf{gw}(K,\mathsf{h}).$$

Theorem 7.5 below shows that this parallel composition is not lock-free (or not even a session) when $\neg\big((\mathsf{G},\mathsf{h}) \leftrightarrow (\mathsf{G}',\mathsf{k})\big)$ and $\vdash \mathcal{M} : \mathsf{G}$ and $\vdash \mathcal{M}' : \mathsf{G}'$.

We proceed by providing an inductive characterisation of the relation of non-compatibility between processes. We first give the definition of a relation between processes (Definition 7.2), and then we show that this relation coincides with the negation of process compatibility (Lemma 7.3).

Definition 7.2 (Non-Compatibility of Processes). *The non-compatibility relation between processes, dubbed $\not\leftrightarrow$, is the symmetric closure of the relation inductively defined as follows.*

$$\frac{P \neq \mathbf{0}}{P \not\leftrightarrow \mathbf{0}}\ [\textsc{nocomp-}\mathbf{0}] \qquad \frac{}{\mathsf{p}?\Lambda \not\leftrightarrow \mathsf{q}?\Lambda'}\ [\textsc{nocomp-i}] \qquad \frac{}{\mathsf{p}!\Lambda \not\leftrightarrow \mathsf{q}!\Lambda'}\ [\textsc{nocomp-o}]$$

$$\frac{msg(\Lambda') \not\subseteq msg(\Lambda)}{\mathsf{p}!\Lambda \not\leftrightarrow \mathsf{q}?\Lambda'}\ [\textsc{nocomp-lab}] \qquad \frac{P \not\leftrightarrow Q}{\mathsf{p}!(\ell.P \uplus \Lambda) \not\leftrightarrow \mathsf{q}?(\ell.Q \uplus \Lambda')}\ [\textsc{nocomp-hered}]$$

Lemma 7.3 (Negation of Compatibility). *$P \not\leftrightarrow Q$ iff $\neg(P \leftrightarrow Q)$.*

Proof. If $P \not\leftrightarrow Q$, then we can show $\neg(P \leftrightarrow Q)$ by induction on the derivation of $P \not\leftrightarrow Q$. We just develop the inductive case [NOCOMP-HERED]. In this case, $P = \mathsf{p}!(\ell.P' \uplus \Lambda)$ and $Q = \mathsf{q}?(\ell.Q' \uplus \Lambda')$; moreover, $P' \not\leftrightarrow Q'$ and thus, by the induction hypothesis, $\neg(P' \leftrightarrow Q')$. We now notice that $P \leftrightarrow Q$ could only possibly hold by rule [COMP-O/I] but, since $\neg(P' \leftrightarrow Q')$, at least one of the coinductive premises of such a rule is not satisfied. Hence, we conclude

$\neg(P \leftrightarrow Q)$.
Vice versa, if $\neg(P \leftrightarrow Q)$ we construct a derivation of $P \not\leftrightarrow Q$ by looking at a "failing derivation" of $P \leftrightarrow Q$. If we try to apply rule [COMP-O/I] to show $P \leftrightarrow Q$, there exists a derivation branch that fails after a finite number, say n, of steps, i.e. that reaches two processes P', Q' which do *not* match the conclusion of [COMP-O/I]. We prove $P \not\leftrightarrow Q$ by induction on n, turning the failing coinductive derivation branch into a derivation of depth $n+1$ which concludes $P \not\leftrightarrow Q$:

- base case $n = 0$. The derivation fails immediately, i.e. $P' = P$ and $Q' = Q$. By cases on the possible shapes of P and Q, we construct a derivation which concludes $P \not\leftrightarrow Q$ in $1 = n + 1$ steps, by one of the axioms [NOCOMP-**0**], [NOCOMP-I], [NOCOMP-O], or [NOCOMP-LAB];

- inductive case $n = m + 1$. The shapes of P, Q match the conclusion of rule [NOCOMP-HERED], but there is some coinductive premise $P' \leftrightarrow Q'$ whose sub-derivation has a branch that fails after m steps. By the induction hypothesis, there exists a derivation of depth $m + 1$ that concludes $P' \not\leftrightarrow Q'$; using this as a premise of rule [NOCOMP-HERED] we construct a derivation of depth $(m + 1) + 1 = n + 1$ which concludes $P \not\leftrightarrow Q$. □

A lemma connecting projections of global types with reductions of typed sessions is handy in the proof of the main result of this section (Theorem 7.5).

Lemma 7.4 (Projection and Reduction). *Let* $\vdash \mathcal{M} : \mathsf{G}$.

(i) If $\mathsf{G}\upharpoonright_{\mathsf{p}} = \mathsf{q}?\Lambda$ *with* $\ell \in msg(\Lambda)$*, then there is a reduction*

$$\mathcal{M} \longrightarrow^* \mathcal{M}' \mid \mathsf{q} \triangleright \mathsf{p}!\Lambda' \text{ such that } \ell \in msg(\Lambda').$$

(ii) If $\mathsf{G}\upharpoonright_{\mathsf{p}} = \mathsf{q}!\Lambda$ *with* $\ell \in msg(\Lambda)$*, then there is a reduction*

$$\mathcal{M} \longrightarrow^* \mathcal{M}' \mid \mathsf{q} \triangleright \mathsf{p}?\Lambda' \text{ such that } \ell \in msg(\Lambda').$$

Proof. The proofs of both items are by induction on $w = depth(\mathsf{G}, \mathsf{p})$.
(i). If $w = 0$, then $\mathsf{G} = \mathsf{q} \to \mathsf{p} : \Gamma$ and $\mathsf{q} \triangleright Q \in \mathcal{M}$ and $Q \leqslant \mathsf{G}\upharpoonright_{\mathsf{q}}$. We get the statement without reducing $\mathcal{M}$, since $\mathsf{G}\upharpoonright_{\mathsf{q}} = \mathsf{p}!\Lambda'$ and $msg(\Lambda') = msg(\Lambda) = msg(\Gamma)$.

If $w > 0$, then $\mathsf{G} = \mathsf{r} \to \mathsf{s} : \{\ell_i.\mathsf{G}_i \mid 1 \leq i \leq n\}$ with $\mathsf{p} \notin \{\mathsf{r}, \mathsf{s}\}$, then by Definition 3.3 there is j, $1 \leq j \leq n$, such that $\mathsf{G}_j\upharpoonright_{\mathsf{p}} = \mathsf{q}?\Lambda_j$ with $\ell \in \Lambda_j$. By rule [ECOMM] of Definition 4.3 $\mathsf{G} \xrightarrow{\mathsf{r}\ell_j\mathsf{s}} \mathsf{G}_j$. By Theorem 4.9(ii) $\mathcal{M} \xrightarrow{\mathsf{r}\ell_j\mathsf{s}} \mathcal{M}''$ and $\vdash \mathcal{M}'' : \mathsf{G}_j$. We conclude using induction, since by Lemma 4.6 $depth(\mathsf{G}_j, \mathsf{p}) < w$.

The proof of (ii) is simpler than the proof of (i), since when $w > 0$ and $\mathsf{G} = \mathsf{r} \to \mathsf{s} : \{\ell_i.\mathsf{G}_i \mid 1 \leq i \leq n\}$ with $\mathsf{p} \notin \{\mathsf{r}, \mathsf{s}\}$, then by Definition 3.3 we have $\mathsf{G}_i\upharpoonright_{\mathsf{p}} = \mathsf{G}_1\upharpoonright_{\mathsf{p}}$ for all i, $1 \leq i \leq n$. □

We can now prove that, whereas compatibility between sessions is not a necessary condition for lock-freedom preservation (as shown in Example 7.1), compatibility between global types is.

Theorem 7.5 (Necessity of Compatibility between Global Types). *If* $\neg\big((\mathsf{G}, \mathsf{h}) \leftrightarrow (\mathsf{G}', \mathsf{k})\big)$ *and* $\vdash \mathcal{M} \mid \mathsf{h} \triangleright H : \mathsf{G}$ *and* $\vdash \mathcal{M}' \mid \mathsf{k} \triangleright K : \mathsf{G}'$*, then either* $\mathcal{M} \mid \mathcal{M}' \mid \mathsf{h} \triangleright \mathsf{gw}(H, \mathsf{k}) \mid \mathsf{k} \triangleright \mathsf{gw}(K, \mathsf{h})$ *is not a session or it is not lock-free.*

Proof. If $\neg\big((\mathsf{G}, \mathsf{h}) \leftrightarrow (\mathsf{G}', \mathsf{k})\big)$ then by defintion of compatible global types either $\mathtt{ptg}(\mathsf{G}) \cap \mathtt{ptg}(\mathsf{G}') \neq \emptyset$ or $\neg(\mathsf{G}\upharpoonright_{\mathsf{h}} \leftrightarrow \mathsf{G}'\upharpoonright_{\mathsf{k}})$. Given that $\vdash \mathcal{M} \mid \mathsf{h} \triangleright H : \mathsf{G}$ and $\vdash \mathcal{M}' \mid \mathsf{k} \triangleright K : \mathsf{G}'$, $\mathtt{ptg}(\mathsf{G}) \subseteq \mathtt{pts}(\mathcal{M} \mid \mathsf{h} \triangleright H)$ and $\mathtt{ptg}(\mathsf{G}') \subseteq \mathtt{pts}(\mathcal{M}' \mid \mathsf{k} \triangleright K)$. Hence, from $\mathtt{ptg}(\mathsf{G}) \cap \mathtt{ptg}(\mathsf{G}') \neq \emptyset$ we get $\mathtt{pts}(\mathcal{M} \mid \mathsf{h} \triangleright H) \cap \mathtt{pts}(\mathcal{M}' \mid \mathsf{k} \triangleright K) \neq \emptyset$. In this case $\mathcal{M} \mid \mathcal{M}' \mid \mathsf{h} \triangleright \mathsf{gw}(H, \mathsf{k}) \mid \mathsf{k} \triangleright \mathsf{gw}(K, \mathsf{h})$ has at least one repeated participant, so it is not a session. Otherwise $\neg(\mathsf{G}\upharpoonright_{\mathsf{h}} \leftrightarrow \mathsf{G}'\upharpoonright_{\mathsf{k}})$, i.e. $\mathsf{G}\upharpoonright_{\mathsf{h}} \not\leftrightarrow \mathsf{G}'\upharpoonright_{\mathsf{k}}$ by Lemma 7.3. We show that $\mathcal{M} \mid \mathcal{M}' \mid \mathsf{h} \triangleright \mathsf{gw}(H, \mathsf{k}) \mid \mathsf{k} \triangleright \mathsf{gw}(K, \mathsf{h})$ is not lock-free by induction on the derivation of $\mathsf{G}\upharpoonright_{\mathsf{h}} \not\leftrightarrow \mathsf{G}'\upharpoonright_{\mathsf{k}}$ (cf. Definition 7.2). In each case we use that $\vdash \mathcal{M} \mid \mathsf{h} \triangleright H : \mathsf{G}$ implies $H \leqslant \mathsf{G}\upharpoonright_{\mathsf{h}}$ and $\vdash \mathcal{M} \mid \mathsf{k} \triangleright K : \mathsf{G}'$ implies $K \leqslant \mathsf{G}'\upharpoonright_{\mathsf{k}}$ by Lemma 4.1.
If $\mathsf{G}\upharpoonright_{\mathsf{h}} = \mathbf{0}$ and $\mathsf{G}'\upharpoonright_{\mathsf{k}} \neq \mathbf{0}$, then $H = \mathbf{0}$ and $K \neq \mathbf{0}$. This implies $\mathsf{gw}(H, \mathsf{k}) = \mathbf{0}$ but $\mathsf{gw}(K, \mathsf{h}) \neq \mathbf{0}$. The messages in $\mathsf{gw}(K, \mathsf{h})$ with sender or receiver h will never be consumed.
If $\mathsf{G}\upharpoonright_{\mathsf{h}} = \mathsf{p}?\Lambda_1$ and $\mathsf{G}'\upharpoonright_{\mathsf{k}} = \mathsf{q}?\Lambda_2$, then $H = \mathsf{p}?\Lambda_1'$ and $K = \mathsf{q}?\Lambda_2'$. This implies $\mathsf{gw}(H, \mathsf{k})$ (after receiving a message from p) wants to send a message to k but $\mathsf{gw}(K, \mathsf{h})$ (after receiving a message from q) wants to send a message to h.
If $\mathsf{G}\upharpoonright_{\mathsf{h}} = \mathsf{p}!\Lambda_1$ and $\mathsf{G}'\upharpoonright_{\mathsf{k}} = \mathsf{q}!\Lambda_2$, then $H = \mathsf{p}!\Lambda_1'$ and $K = \mathsf{q}!\Lambda_2'$. This implies $\mathsf{gw}(H, \mathsf{k})$ (before sending a message to p) waits for a message from k but $\mathsf{gw}(K, \mathsf{h})$ (before sending a message to q) waits for a message from h.
Let $\mathsf{G}\upharpoonright_{\mathsf{h}} = \mathsf{p}!\Lambda_1$ and $\mathsf{G}'\upharpoonright_{\mathsf{k}} = \mathsf{q}?(\ell.P \uplus \Lambda_2)$ and $\ell \notin msg(\Lambda_1)$, then $H = \mathsf{p}!\Lambda_1'$ with $msg(\Lambda_1') = msg(\Lambda_1)$ and $K = \mathsf{q}?(\ell.P' \uplus \Lambda_2')$ with $msg(\Lambda_2') \supseteq msg(\Lambda_2)$. From $\mathsf{G}'\upharpoonright_{\mathsf{k}} = \mathsf{q}?(\ell.P \uplus \Lambda_2)$ we get $\mathcal{M}' \longrightarrow^* \mathcal{M}_1' \mid \mathsf{q} \triangleright Q$ with $Q = \mathsf{k}!(\ell.Q' \uplus \Lambda_3)$ by

Lemma 7.4(i). After Q exchanges the message ℓ with $\mathsf{gw}(K,\mathsf{h})$, the process $\mathsf{gw}(K,\mathsf{h})$ wants to send the message ℓ to h, but $\mathsf{gw}(H,\mathsf{k})$ cannot receive this message. More precisely

$$\begin{array}{l}\mathcal{M} \mid \mathcal{M}' \mid \mathsf{h} \triangleright \mathsf{gw}(H,\mathsf{k}) \mid \mathsf{k} \triangleright \mathsf{gw}(K,\mathsf{h}) \longrightarrow^* \\ \quad \mathcal{M} \mid \mathcal{M}'_1 \mid \mathsf{q} \triangleright Q \mid \mathsf{h} \triangleright \mathsf{gw}(H,\mathsf{k}) \mid \mathsf{k} \triangleright \mathsf{gw}(K,\mathsf{h}) \xrightarrow{\mathsf{q}\ell\mathsf{k}} \\ \quad\quad \mathcal{M} \mid \mathcal{M}'_1 \mid \mathsf{q} \triangleright Q' \mid \mathsf{h} \triangleright \mathsf{gw}(H,\mathsf{k}) \mid \mathsf{k} \triangleright \mathsf{h}!\ell.\mathsf{gw}(P,\mathsf{h})\end{array}$$

which cannot reduce since $\mathsf{gw}(H,\mathsf{k}) = \mathsf{k}?\Lambda$ with $\ell \notin msg(\Lambda) = msg(\Lambda_1)$. Let $\mathsf{G}\upharpoonright_{\mathsf{h}} = \mathsf{p}!(\ell.\mathsf{G}_1\upharpoonright_{\mathsf{h}} \uplus \Lambda_1)$ and $\mathsf{G}'\upharpoonright_{\mathsf{k}} = \mathsf{q}?(\ell.\mathsf{G}_2\upharpoonright_{\mathsf{k}} \uplus \Lambda_2)$ and $\mathsf{G}_1\upharpoonright_{\mathsf{h}} \not\twoheadrightarrow \mathsf{G}_2\upharpoonright_{\mathsf{k}}$. Then $H = \mathsf{p}!(\ell.P \uplus \Lambda'_1)$ and $K = \mathsf{q}?(\ell.Q \uplus \Lambda'_2)$ and $P \leqslant \mathsf{G}_1\upharpoonright_{\mathsf{h}}$, $Q \leqslant \mathsf{G}_2\upharpoonright_{\mathsf{k}}$. By Lemma 7.4 $\mathcal{M} \longrightarrow^* \mathcal{M}_1 \mid \mathsf{p} \triangleright \mathsf{h}?(\ell.P' \uplus \Lambda_3)$ and $\mathcal{M}' \longrightarrow^* \mathcal{M}'_1 \mid \mathsf{q} \triangleright \mathsf{k}!(\ell.Q' \uplus \Lambda'_3)$. Then

$$\mathcal{M} \mid \mathsf{h} \triangleright H \longrightarrow^* \mathcal{M}_1 \mid \mathsf{p} \triangleright \mathsf{h}?(\ell.P' \uplus \Lambda_3) \mid \mathsf{h} \triangleright H \xrightarrow{\mathsf{h}\ell\mathsf{p}} \mathcal{M}_1 \mid \mathsf{p} \triangleright P' \mid \mathsf{h} \triangleright P$$

and by the Subject Reduction Theorem (Theorem 4.8) there is G'_1 such that $\vdash \mathcal{M}_1 \mid \mathsf{p} \triangleright P' \mid \mathsf{h} \triangleright P : \mathsf{G}'_1$. Similarly

$$\mathcal{M}' \mid \mathsf{k} \triangleright K \longrightarrow^* \mathcal{M}'_1 \mid \mathsf{q} \triangleright \mathsf{k}!(\ell.Q' \uplus \Lambda'_3) \mid \mathsf{k} \triangleright K \xrightarrow{\mathsf{q}\ell\mathsf{k}} \mathcal{M}'_1 \mid \mathsf{q} \triangleright Q' \mid \mathsf{k} \triangleright Q$$

and $\vdash \mathcal{M}'_1 \mid \mathsf{q} \triangleright Q' \mid \mathsf{k} \triangleright Q : \mathsf{G}'_2$ for some G'_2. Building G'_1 and G'_2 as in the proof of Theorem 4.8 we get $\mathsf{G}'_1\upharpoonright_{\mathsf{h}} = \mathsf{G}_1\upharpoonright_{\mathsf{h}}$ and $\mathsf{G}'_2\upharpoonright_{\mathsf{k}} = \mathsf{G}_2\upharpoonright_{\mathsf{k}}$, which imply $\mathsf{G}'_1\upharpoonright_{\mathsf{h}} \not\twoheadrightarrow \mathsf{G}'_2\upharpoonright_{\mathsf{k}}$. When $\mathsf{gw}(K,\mathsf{h})$ and $\mathsf{gw}(H,\mathsf{k})$ exchange the message ℓ they become $\mathsf{gw}(P,\mathsf{k})$ and $\mathsf{gw}(Q,\mathsf{h})$, so the network is not lock-free by induction hypothesis. More precisely

$$\begin{array}{l}\mathcal{M} \mid \mathcal{M}' \mid \mathsf{h} \triangleright \mathsf{gw}(H,\mathsf{k}) \mid \mathsf{k} \triangleright \mathsf{gw}(K,\mathsf{h}) \longrightarrow^* \\ \mathcal{M} \mid \mathcal{M}'_1 \mid \mathsf{q} \triangleright \mathsf{k}!(\ell.Q' \uplus \Lambda'_3) \mid \mathsf{h} \triangleright \mathsf{gw}(H,\mathsf{k}) \mid \mathsf{k} \triangleright \mathsf{gw}(K,\mathsf{h}) \xrightarrow{\mathsf{q}\ell\mathsf{k}} \\ \mathcal{M} \mid \mathcal{M}'_1 \mid \mathsf{q} \triangleright Q' \mid \mathsf{h} \triangleright \mathsf{gw}(H,\mathsf{k}) \mid \mathsf{k} \triangleright \mathsf{h}!\ell.\mathsf{gw}(Q,\mathsf{h}) \xrightarrow{\mathsf{k}\ell\mathsf{h}} \\ \mathcal{M} \mid \mathcal{M}'_1 \mid \mathsf{q} \triangleright Q' \mid \mathsf{h} \triangleright \mathsf{p}!\ell.\mathsf{gw}(P,\mathsf{k}) \mid \mathsf{k} \triangleright \mathsf{gw}(Q,\mathsf{h}) \longrightarrow^* \\ \mathcal{M}_1 \mid \mathsf{p} \triangleright \mathsf{h}?(\ell.P' \uplus \Lambda_3) \mid \mathcal{M}'_1 \mid \mathsf{q} \triangleright Q' \mid \mathsf{h} \triangleright \mathsf{p}!\ell.\mathsf{gw}(P,\mathsf{k}) \mid \mathsf{k} \triangleright \mathsf{gw}(Q,\mathsf{h}) \xrightarrow{\mathsf{h}\ell\mathsf{p}} \\ \mathcal{M}_1 \mid \mathsf{p} \triangleright P' \mid \mathcal{M}'_1 \mid \mathsf{q} \triangleright Q' \mid \mathsf{h} \triangleright \mathsf{gw}(P,\mathsf{k}) \mid \mathsf{k} \triangleright \mathsf{gw}(Q,\mathsf{h})\end{array}$$

which is not lock-free by induction hypothesis. □

Notice that Theorem 7.5 above is stated for typeable multiparty sessions, since it uses compatibility of their global types.

8. Direct Composition of Typed Multiparty Sessions

The use of gateways enables us to get a "safe" composition of systems by modifying just the interface participants. Therefore such an approach is useful after implementation, since it minimally affects systems that are already implemented. On the other hand, one would prefer to avoid the overhead due to gateways when composition is performed during development, as part of a modular design of systems via global types. In this setting one could easily modify the behaviour of any participant, since they have not been implemented yet.

One could hence wonder whether gateways are strictly necessary to get safe compositions in our multiparty-sessions setting. One could quite naturally expect that it is possible to "bypass" the use of gateways by removing them and simply applying a renaming function on the other participants' "communication behaviour". This however would be too naive an approach. As pointed out in [2] (Sect. 5.3), a "safe" composition of the systems in [2] which bypasses the use of gateways could require an heavy redesign of the participants involved in the inter-systems interactions. A disciplined setting as the present one, instead, enables direct composition to be handled more easily, both at the level of sessions and of global types, but some care is anyway needed. The following example, in fact, shows that just a renaming would not work in general.

Example 8.1 (Renaming is not Enough for Direct Composition). Consider the following global types and multiparty sessions.

$$\begin{array}{rclcrcl} \mathsf{G} & = & \mathsf{p} \to \mathsf{h} : \ell.\mathsf{G} & \qquad & \mathsf{G}' & = & \mathsf{k} \to \mathsf{r} : \ell.\mathsf{k} \to \mathsf{s} : \ell.\mathsf{G}' \\ \mathcal{M} & = & \mathsf{p} \triangleright P \mid \mathsf{h} \triangleright H & & \mathcal{M}' & = & \mathsf{r} \triangleright R \mid \mathsf{s} \triangleright S \mid \mathsf{k} \triangleright K \end{array}$$

where

$$P = \mathsf{h}!\ell.P \qquad H = \mathsf{p}?\ell.H \qquad R = \mathsf{k}?\ell.R \qquad S = \mathsf{k}?\ell.S \qquad K = \mathsf{r}!\ell.\mathsf{s}!\ell.K$$

It is easy to check that $\vdash \mathcal{M} : \mathsf{G}$ and $\vdash \mathcal{M}' : \mathsf{G}'$.

In order to compose the above multiparty sessions, bypassing the gateways, we could take K out on the side of $\mathcal{M}'$, and rename some senders' and receivers' names in R and S so that they can receive the message ℓ directly from p, obtaining $\tilde{R} = \mathsf{p}?\ell.\tilde{R}$ and $\tilde{S} = \mathsf{p}?\ell.\tilde{S}$. On the side of $\mathcal{M}$, instead, after taking out H, we could not get a sound composition by a simple renaming for the recipient h in $P = \mathsf{h}!\ell.P$, since the message ℓ should be delivered,

alternately, to $\tilde{R}$ and $\tilde{S}$. A safe direct composition would hence imply a less straightforward modification of P into $\tilde{P} = \mathsf{r}!\ell.\mathsf{s}!\ell.\tilde{P}$. $\diamond$

We implement a direct composition at the level of global types by means of the function $\_ \,{}^{\mathsf{h}\bowtie\mathsf{k}}\, \_$ defined below and we shall then use this function to achieve a similar composition of multiparty sessions as well. The composition of a communication between p as sender and h as receiver with a communication between k as sender and s as receiver gives a communication between p as sender and s as receiver. Similarly when the roles of h and k are exchanged. Instead the communications not involving h and k remain unchanged.

Definition 8.2 (Global Type Direct Composition). *Let* $(\mathsf{G}, \mathsf{h}) \leftrightarrow (\mathsf{G}', \mathsf{k})$. *We define*

$$\mathsf{G} \,{}^{\mathsf{h}\bowtie\mathsf{k}}\, \mathsf{G}'$$

coinductively by the clauses of Figure 11, assuming $\{\mathsf{p}, \mathsf{q}, \mathsf{r}, \mathsf{s}\} \cap \{\mathsf{h}, \mathsf{k}\} = \emptyset$. *The clauses must be applied in the given order.*

---

(1) $\mathtt{End} \,{}^{\mathsf{h}\bowtie\mathsf{k}}\, \mathsf{G} = \mathsf{G}$

(2) $\mathsf{p} \to \mathsf{h} : \{\ell_i.\mathsf{G}_i \mid {\scriptstyle 1\le i\le n}\} \,{}^{\mathsf{h}\bowtie\mathsf{k}}\, \mathsf{k} \to \mathsf{s} : \{\ell_i.\mathsf{G}'_i \mid {\scriptstyle 1\le i\le n}\} \uplus \Gamma = \mathsf{p} \to \mathsf{s} : \{\ell_i.\mathsf{G}_i \,{}^{\mathsf{h}\bowtie\mathsf{k}}\, \mathsf{G}'_i \mid {\scriptstyle 1\le i\le n}\}$

(3) $\mathsf{h} \to \mathsf{p} : \{\ell_i.\mathsf{G}_i \mid {\scriptstyle 1\le i\le n}\} \uplus \Gamma \,{}^{\mathsf{h}\bowtie\mathsf{k}}\, \mathsf{s} \to \mathsf{k} : \{\ell_i.\mathsf{G}'_i \mid {\scriptstyle 1\le i\le n}\} = \mathsf{s} \to \mathsf{p} : \{\ell_i.\mathsf{G}_i \,{}^{\mathsf{h}\bowtie\mathsf{k}}\, \mathsf{G}'_i \mid {\scriptstyle 1\le i\le n}\}$

(4) $\mathsf{p} \to \mathsf{q} : \{\ell_i.\mathsf{G}_i \mid {\scriptstyle 1\le i\le n}\} \,{}^{\mathsf{h}\bowtie\mathsf{k}}\, \mathsf{G} = \mathsf{p} \to \mathsf{q} : \{\ell_i.\mathsf{G} \,{}^{\mathsf{k}\bowtie\mathsf{h}}\, \mathsf{G}_i \mid {\scriptstyle 1\le i\le n}\}$

(5) $\mathsf{G} \,{}^{\mathsf{h}\bowtie\mathsf{k}}\, \mathsf{r} \to \mathsf{s} : \{\ell_i.\mathsf{G}_i \mid {\scriptstyle 1\le i\le n}\} = \mathsf{r} \to \mathsf{s} : \{\ell_i.\mathsf{G}_i \,{}^{\mathsf{k}\bowtie\mathsf{h}}\, \mathsf{G} \mid {\scriptstyle 1\le i\le n}\}$

Figure 11: Definition of $\_ \,{}^{\mathsf{h}\bowtie\mathsf{k}}\, \_$ on global types

The rules are in direct correspondence with the rules of composition via gateways (Definition 6.4) and the intuition is the same.

Taking G and G' of Example 8.1, it is easy to check that $(\mathsf{G}, \mathsf{h}) \leftrightarrow (\mathsf{G}', \mathsf{k})$ and hence that we can apply direct composition on these global types, obtaining:

$$\mathsf{G} \,{}^{\mathsf{h}\bowtie\mathsf{k}}\, \mathsf{G}' \quad = \quad \mathsf{p} \to \mathsf{r} : \ell.\mathsf{p} \to \mathsf{s} : \ell.(\mathsf{G} \,{}^{\mathsf{h}\bowtie\mathsf{k}}\, \mathsf{G}')$$

We prove now the soundness of the direct composition definition.

Lemma 8.3 (Direct Composition is Well-Defined). *Let* $(\mathsf{G},\mathsf{h}) \leftrightarrow (\mathsf{G}',\mathsf{k})$. *Then* $\mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}'$ *is defined and it is a global type.*

Proof. This lemma is the analogous of Lemma 6.8. Notice that Figure 9 and Figure 11 have exactly the same global types on the left side of the equalities. The proof of Lemma 6.8 only uses these global types. Therefore we can easily adapt these proofs to $\mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}'$. □

The global type $\mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}'$ can be shown to be well-formed and to preserve the projections of those participants which do not communicate with h or k.

Theorem 8.4 (Properties of Direct Composition). *If* $(\mathsf{G},\mathsf{h}) \leftrightarrow (\mathsf{G}',\mathsf{k})$, *then* $\mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}'$ *is well-formed. If* $\mathsf{p} \in \mathtt{ptg}(\mathsf{G})$ *and* $\mathsf{h} \notin \mathtt{ptp}(\mathsf{G}\upharpoonright_{\mathsf{p}})$, *then* $\mathsf{G}\upharpoonright_{\mathsf{p}} = (\mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}')\upharpoonright_{\mathsf{p}}$. *If* $\mathsf{p} \in \mathtt{ptg}(\mathsf{G}')$ *and* $\mathsf{k} \notin \mathtt{ptp}(\mathsf{G}'\upharpoonright_{\mathsf{p}})$, *then* $\mathsf{G}'\upharpoonright_{\mathsf{p}} = (\mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}')\upharpoonright_{\mathsf{p}}$.

Proof. To show well-formedness of $\mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}'$ we have to show that each participant has finite depth in each subterm of $\mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}'$ and that $\mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}'$ is projectable. It is easy to verify that if

$$w = max\{depth(\mathsf{G}_0,\mathsf{p}) \mid \mathsf{G}_0 \text{ is a subterm of } \mathsf{G} \text{ and } \mathsf{p} \in \mathtt{ptg}(\mathsf{G})\} \text{ and}$$
$$w' = max\{depth(\mathsf{G}'_0,\mathsf{p}) \mid \mathsf{G}'_0 \text{ is a subterm of } \mathsf{G}' \text{ and } \mathsf{p} \in \mathtt{ptg}(\mathsf{G}')\}$$

then $depth(\hat{\mathsf{G}},\mathsf{p}) \leq max\{w,w'\}$ for all $\hat{\mathsf{G}}$ subterm of $\mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}'$ and all $\mathsf{p} \in \mathtt{ptg}(\mathsf{G}) \cup \mathtt{ptg}(\mathsf{G}')$.

We can show that $\mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}'$ is projectable using the projectability of G,G' and the definition of $\mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}'$.
Let us now move to the second property. If $\mathsf{p} \in \mathtt{ptg}(\mathsf{Y})$ and $\mathsf{h} \notin \mathtt{ptp}(\mathsf{Y}\upharpoonright_{\mathsf{p}})$ we prove that $\mathsf{Y}\upharpoonright_{\mathsf{p}} = (\mathsf{Y} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{Y}')\upharpoonright_{\mathsf{p}}$ for any recursive call $\mathsf{Y} \,{}^{\mathsf{h}\not\leftrightarrow\mathsf{k}}\, \mathsf{Y}'$ in $\mathsf{G} \,{}^{\mathsf{h}\not\leftrightarrow\mathsf{k}}\, \mathsf{G}'$ by coinduction and by cases on the last applied rule. Since $\mathsf{h} \notin \mathtt{ptp}(\mathsf{Y}\upharpoonright_{\mathsf{p}})$ only rules (4) and (5) can be used and these rules do not modify the communications of p, so we get the statement. The case of $\mathsf{p} \in \mathtt{ptg}(\mathsf{Y}')$ and $\mathsf{k} \notin \mathtt{ptp}(\mathsf{Y}'\upharpoonright_{\mathsf{p}})$ is analogous. □

We can hence obtain a direct composition function at the multiparty-session level through the use of the function $\_ \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \_$ on global types.

Definition 8.5 (Multiparty Session Direct Composition).
Let $\vdash \mathcal{M} : \mathsf{G}$, $\vdash \mathcal{M}' : \mathsf{G}'$, $(\mathcal{M},\mathsf{h}) \leftrightarrow (\mathcal{M}',\mathsf{k})$, *with* $\mathcal{M} \equiv \mathcal{M}_1 \mid \mathcal{M}_2$ *and* $\mathcal{M}' \equiv \mathcal{M}'_1 \mid \mathcal{M}'_2$ *be such that:*

- $\mathsf{h} \in \mathtt{ptp}(P)$ *for all* $\mathsf{p} \triangleright P \in \mathcal{M}_1$
- $\mathsf{h} \notin \mathtt{ptp}(P)$ *for all* $\mathsf{p} \triangleright P \in \mathcal{M}_2$
- $\mathsf{k} \in \mathtt{ptp}(Q)$ *for all* $\mathsf{q} \triangleright Q \in \mathcal{M}'_1$
- $\mathsf{k} \notin \mathtt{ptp}(Q)$ *for all* $\mathsf{q} \triangleright Q \in \mathcal{M}'_2$

We define

$$\mathcal{M} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathcal{M}' = \prod_{\mathsf{p}\in\mathtt{pts}(\mathcal{M}_1)} \mathsf{p} \triangleright (\mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}')\upharpoonright_{\mathsf{p}} \;\mid \mathcal{M}_2 \mid \prod_{\mathsf{q}\in\mathtt{pts}(\mathcal{M}'_1)} \mathsf{q} \triangleright (\mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}')\upharpoonright_{\mathsf{q}} \;\mid \mathcal{M}'_2$$

Intuitively, participants not interacting with the interface are left unchanged, while the behaviour of the others is extracted from the composed global type.

The main result of this section connects the direct composition of global types and the direct composition of multiparty sessions.

Theorem 8.6 (Typing Direct Composition). *If* $(\mathcal{M}, \mathsf{h}) \leftrightarrow (\mathcal{M}', \mathsf{k})$ *and* $\vdash \mathcal{M} : \mathsf{G}$ *and* $\vdash \mathcal{M}' : \mathsf{G}'$*, then* $\vdash \mathcal{M} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathcal{M}' : \mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}'$.

Proof. Lemma 6.2 gives $(\mathsf{G}, \mathsf{h}) \leftrightarrow (\mathsf{G}', \mathsf{k})$. By Theorem 8.4 $\mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}'$ is well-formed. We conclude $\vdash \mathcal{M} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathcal{M}' : \mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}'$ using the relations on the projections of $\mathsf{G} \,{}^{\mathsf{h}\!\leftrightarrow\!\mathsf{k}}\, \mathsf{G}'$ given in Theorem 8.4. □

Example 8.7. It is almost immediate to check that $(\mathcal{M}, \mathsf{h}) \leftrightarrow (\mathcal{M}', \mathsf{k})$ for Example 8.1. We obtain

$$\mathcal{M} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathcal{M}' = \mathsf{p} \triangleright \tilde{P} \mid \mathsf{r} \triangleright \tilde{R} \mid \mathsf{s} \triangleright \tilde{S}$$

where

$$\tilde{R} = \mathsf{p}?\ell.\tilde{R} \qquad \tilde{S} = \mathsf{p}?\ell.\tilde{S} \qquad \tilde{P} = \mathsf{r}!\ell.\mathsf{s}!\ell.\tilde{P}$$

and $\vdash \mathcal{M} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathcal{M}' : \mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}'$. ◇

For direct compositions we cannot compose $\mathcal{M}$ and $\mathcal{M}'$ when $\vdash \mathcal{M} : \mathsf{G}$ and $\vdash \mathcal{M}' : \mathsf{G}'$ and $\neg((\mathsf{G}, \mathsf{h}) \leftrightarrow (\mathsf{G}', \mathsf{k}))$, since $\mathsf{G} \,{}^{\mathsf{h}\triangleright\!\triangleleft\mathsf{k}}\, \mathsf{G}'$ is undefined. So we cannot have a result similar to Theorem 7.5.

9. Decomposition of Typed Multiparty Sessions

In order to use direct composition in the modular development of systems one needs first to decompose a global specification into modules, then to develop the various modules in isolation, and finally to combine the modules using direct composition. We still miss an operation to decompose a global specification: this will be introduced in the present section. In particular, we will define a function enabling the decomposition of a global type into two global types, whose direct composition represents the same system of processes typed by the original global type. We start by defining the projection of a global type with respect to a set of participants $\mathcal{P}$ and a new participant h. In this projection:

- a communication between participants in $\mathcal{P}$ remains unchanged;
- a communication between one participant in $\mathcal{P}$ and one participant not in $\mathcal{P}$ becomes a communication between the participant in $\mathcal{P}$ and h;
- a communication between participants not in $\mathcal{P}$ is erased.

Definition 9.1 (Type Projection for Decomposition). *The projection of a global type* G *with respect to a set of participants* $\mathcal{P}$ *and one interface participant* $\mathsf{h} \notin \mathtt{ptg}(\mathsf{G}) \cup \mathcal{P}$ *is the global type* $\mathsf{G}{\Uparrow}^{\mathsf{h}}_{\mathcal{P}}$ *(if any) defined in Figure 12.*

$$\mathsf{G} \Uparrow^{\mathsf{h}}_{\mathcal{P}} = \mathtt{End} \qquad \text{if } \mathtt{ptg}(\mathsf{G}) \cap \mathcal{P} = \emptyset$$

$$(\mathsf{p} \to \mathsf{q} : \Gamma)\Uparrow^{\mathsf{h}}_{\mathcal{P}} = \begin{cases} \mathsf{p} \to \mathsf{q} : \{\ell.\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}} \mid \ell.\mathsf{G} \in \Gamma\} & \text{if } \mathsf{p},\mathsf{q} \in \mathcal{P} \\ \mathsf{p} \to \mathsf{h} : \{\ell.\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}} \mid \ell.\mathsf{G} \in \Gamma\} & \text{if } \mathsf{p} \in \mathcal{P}, \mathsf{q} \notin \mathcal{P} \\ \mathsf{h} \to \mathsf{q} : \{\ell.\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}} \mid \ell.\mathsf{G} \in \Gamma\} & \text{if } \mathsf{p} \notin \mathcal{P}, \mathsf{q} \in \mathcal{P} \\ \widehat{\mathsf{G}} & \text{if } \{\mathsf{p},\mathsf{q}\} \cap \mathcal{P} = \emptyset \text{ and} \\ & \widehat{\mathsf{G}} = \mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}} \text{ for all } \ell.\mathsf{G} \in \Gamma \end{cases}$$

Figure 12: Definition of $\Uparrow$ for global types

It is easy to check that if $\mathtt{ptg}(\mathsf{G}) = \mathcal{P}$, then $\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}} = \mathsf{G}$. Moreover if $\mathcal{P} = \emptyset$, then $\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}} = \mathtt{End}$. For example,

$$(\mathsf{p} \to \mathsf{q} : \ell.\mathsf{r} \to \mathsf{s}.\ell')\Uparrow^{\mathsf{h}}_{\{\mathsf{p},\mathsf{q}\}} = \mathsf{p} \to \mathsf{q} : \ell.(\mathsf{r} \to \mathsf{s}.\ell')\Uparrow^{\mathsf{h}}_{\{\mathsf{p},\mathsf{q}\}} = \mathsf{p} \to \mathsf{q} : \ell.\mathtt{End}$$

Clearly the projection defined above is a partial function. In fact the condition in case $\mathsf{p}, \mathsf{q} \notin \mathcal{P}$ is quite demanding. It requires the projection of a choice between participants not in $\mathcal{P}$ to be the same for all participants in $\mathcal{P}$ (this is a condition used in the standard projection of global types as well, see, e.g. [31, 32]), as shown by the result below.

Proposition 9.2 (Projection Properties). *If* $\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}}$ *is defined and* G *contains* $\mathsf{p} \to \mathsf{q} : \{\ell_i.\mathsf{G}_i \mid {}_{1 \leq i \leq n}\}$ *with* $\{\mathsf{p}, \mathsf{q}\} \cap \mathcal{P} = \emptyset$*, then* $\mathsf{G}_i \upharpoonright_{\mathsf{r}} = \mathsf{G}_1 \upharpoonright_{\mathsf{r}}$ *for all* i*,* $1 \leq i \leq n$*, and for all* $\mathsf{r} \in \mathcal{P}$*.*

Proof. By coinduction on G and by cases on the definition of type projection for decomposition (Definition 9.1). □

The condition above could be relaxed (similarly to what is done in the definition of projection on a single participant, Definition 3.3), but this would complicate the definition and we preferred simplicity.

We can now define the decomposition function on global types. Basically, the participants are split in two different global types, and two fresh interface participants are created to manage the communications between the two groups.

Definition 9.3 (Global Type Decomposition). *Let* G *be a global type,* $\{\mathcal{P}, \mathcal{Q}\}$ *a partition of* $\mathtt{ptg}(\mathsf{G})$ *and* $\mathsf{h}, \mathsf{k} \notin \mathtt{ptg}(\mathsf{G})$ *with* $\mathsf{h} \neq \mathsf{k}$*. We set*

$$\textsc{Dec}^{\mathsf{h},\mathsf{k}}(\mathsf{G}, \mathcal{P}, \mathcal{Q}) = \left(\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}}, \mathsf{G}\Uparrow^{\mathsf{k}}_{\mathcal{Q}}\right)$$

if $\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}}$ *and* $\mathsf{G}\Uparrow^{\mathsf{k}}_{\mathcal{Q}}$ *are defined.*

A first lemma proves that the global types obtained by decomposition are compatible via the fresh participants used in the decomposition.

Lemma 9.4 (Decomposition and Compatibility). *If* $\textsc{Dec}^{\mathsf{h},\mathsf{k}}(\mathsf{G}, \mathcal{P}, \mathcal{Q}) = \left(\mathsf{G}_1, \mathsf{G}_2\right)$*, then* $(\mathsf{G}_1, \mathsf{h}) \leftrightarrow (\mathsf{G}_2, \mathsf{k})$*.*

Proof. Clearly $\mathtt{ptg}(\mathsf{G}_1) \cap \mathtt{ptg}(\mathsf{G}_2) = \emptyset$. By the agreement and compatibility Lemma (6.7) it is enough to show $(\mathsf{G}_1, \mathsf{h}) \leftrightsquigarrow (\mathsf{G}_2, \mathsf{k})$. The proof is by coinduction on G and by cases on Definition 9.1.

If $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \Gamma$ with $\mathsf{p} \in \mathcal{P}$ and $\mathsf{q} \in \mathcal{Q}$, then by Definition 9.3

$$\begin{aligned}
\mathsf{G}_1 &= \mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}} = \mathsf{p} \to \mathsf{h} : \{\ell.\mathsf{G}^{\ell}\Uparrow^{\mathsf{h}}_{\mathcal{P}} \mid \ell.\mathsf{G}^{\ell} \in \Gamma\} \quad \text{and} \\
\mathsf{G}_2 &= \mathsf{G}\Uparrow^{\mathsf{k}}_{\mathcal{Q}} = \mathsf{k} \to \mathsf{q} : \{\ell.\mathsf{G}^{\ell}\Uparrow^{\mathsf{k}}_{\mathcal{Q}} \mid \ell.\mathsf{G}^{\ell} \in \Gamma\}
\end{aligned}$$

By coinduction we have that, for all $\ell.\mathsf{G}^\ell \in \Gamma$, $(\mathsf{G}^\ell\Uparrow^{\mathsf{h}}_{\mathcal{P}}, \mathsf{h}) \leftrightsquigarrow (\mathsf{G}^\ell\Uparrow^{\mathsf{k}}_{\mathcal{Q}}, \mathsf{k})$ and hence we can derive $(\mathsf{G}_1, \mathsf{h}) \leftrightsquigarrow (\mathsf{G}_2, \mathsf{k})$ using rule [REL-COMM] of the definition of the agreement relation (Definition 6.6).

If $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \Gamma$ with $\mathsf{p}, \mathsf{q} \in \mathcal{P}$, then

$$\mathsf{G}_1 = \mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}} = \mathsf{p} \to \mathsf{q} : \{\ell.\mathsf{G}^\ell\Uparrow^{\mathsf{h}}_{\mathcal{P}} |\ \ell.\mathsf{G}^\ell \in \Gamma\} \quad \text{and} \quad \mathsf{G}_2 = \mathsf{G}\Uparrow^{\mathsf{k}}_{\mathcal{Q}} = \widehat{\mathsf{G}}$$

for some global type $\widehat{\mathsf{G}}$ such that $\widehat{\mathsf{G}} = \mathsf{G}^\ell\Uparrow^{\mathsf{k}}_{\mathcal{Q}}$ for all $\ell.\mathsf{G}^\ell \in \Gamma$. By coinduction we have that $(\mathsf{G}^\ell\Uparrow^{\mathsf{h}}_{\mathcal{P}}, \mathsf{h}) \leftrightsquigarrow (\mathsf{G}^\ell\Uparrow^{\mathsf{k}}_{\mathcal{Q}}, \mathsf{k})$ for all $\ell.\mathsf{G}^\ell \in \Gamma$. Then $(\mathsf{G}_1, \mathsf{h}) \leftrightsquigarrow (\mathsf{G}_2, \mathsf{k})$ using rule [REL-GO] of Definition 6.6. □

The following theorem shows that decomposition is the left inverse of direct composition.

Theorem 9.5 (Left Inverse). *If* $\textsc{Dec}^{\mathsf{h},\mathsf{k}}(\mathsf{G}, \mathcal{P}, \mathcal{Q}) = \big(\mathsf{G}_1, \mathsf{G}_2\big)$*, then*

$$\mathsf{G}_1 \ {}^{\mathsf{h}\bowtie\mathsf{k}}\ \mathsf{G}_2 = \mathsf{G}.$$

Proof. We show the thesis by coinduction on the structure of G and by cases on the definition of projection for decomposition (Definition 9.1).

If $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \Gamma$ with $\mathsf{p} \in \mathcal{P}$ and $\mathsf{q} \in \mathcal{Q}$ then

$$\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}} = \mathsf{p} \to \mathsf{h} : \{\ell.\mathsf{G}^\ell\Uparrow^{\mathsf{h}}_{\mathcal{P}} |\ \ell.\mathsf{G}^\ell \in \Gamma\} \quad \text{and} \quad \mathsf{G}\Uparrow^{\mathsf{k}}_{\mathcal{Q}} = \mathsf{k} \to \mathsf{q} : \{\ell.\mathsf{G}^\ell\Uparrow^{\mathsf{k}}_{\mathcal{Q}} |\ \ell.\mathsf{G}^\ell \in \Gamma\}$$

By definition of direct composition of global types (Definition 8.2), we get

$$\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}}{}^{\mathsf{h}\bowtie\mathsf{k}}\ \mathsf{G}\Uparrow^{\mathsf{k}}_{\mathcal{Q}} = \mathsf{p} \to \mathsf{q} : \{\ell.(\mathsf{G}^\ell\Uparrow^{\mathsf{h}}_{\mathcal{P}}{}^{\mathsf{h}\bowtie\mathsf{k}}\ \mathsf{G}^\ell\Uparrow^{\mathsf{k}}_{\mathcal{Q}}) \mid \ell.\mathsf{G}^\ell \in \Gamma\}$$

By coinduction we have that $\mathsf{G}^\ell = \mathsf{G}^\ell\Uparrow^{\mathsf{h}}_{\mathcal{P}}{}^{\mathsf{h}\bowtie\mathsf{k}}\ \mathsf{G}^\ell\Uparrow^{\mathsf{k}}_{\mathcal{Q}}$ for all $\ell.\mathsf{G}^\ell \in \Gamma$ and hence the thesis.

If $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \Gamma$ with $\mathsf{p}, \mathsf{q} \in \mathcal{P}$ then

$$\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}} = \mathsf{p} \to \mathsf{q} : \{\ell.\mathsf{G}^\ell\Uparrow^{\mathsf{h}}_{\mathcal{P}} |\ {}_{1\le i\le n}\} \quad \text{and} \quad \mathsf{G}\Uparrow^{\mathsf{k}}_{\mathcal{Q}} = \widehat{\mathsf{G}}$$

for some global type $\widehat{\mathsf{G}}$ such that $\widehat{\mathsf{G}} = \mathsf{G}^\ell\Uparrow^{\mathsf{k}}_{\mathcal{Q}}$ for all $\ell.\mathsf{G}^\ell \in \Gamma$. By Definition 8.2 we get

$$\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}}{}^{\mathsf{h}\bowtie\mathsf{k}}\ \mathsf{G}\Uparrow^{\mathsf{k}}_{\mathcal{Q}} = \mathsf{p} \to \mathsf{q} : \{\ell.(\mathsf{G}^\ell\Uparrow^{\mathsf{h}}_{\mathcal{P}}{}^{\mathsf{h}\bowtie\mathsf{k}}\ \widehat{\mathsf{G}}) \mid \ell.\mathsf{G}^\ell \in \Gamma\}$$

which implies

$$\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}}{}^{\mathsf{h}\bowtie\mathsf{k}}\ \mathsf{G}\Uparrow^{\mathsf{k}}_{\mathcal{Q}} = \mathsf{p} \to \mathsf{q} : \{\ell.(\mathsf{G}^\ell\Uparrow^{\mathsf{h}}_{\mathcal{P}}{}^{\mathsf{h}\bowtie\mathsf{k}}\ \mathsf{G}^\ell\Uparrow^{\mathsf{k}}_{\mathcal{Q}}) \mid \ell.\mathsf{G}^\ell \in \Gamma\}$$

and we can conclude as in previous case. □

Decomposition instead is not the right inverse of direct composition, since in building $\mathsf{G}_1 \ {}^{\mathsf{h}}\!\bowtie^{\mathsf{k}} \mathsf{G}_2$ we discharge alternatives which cannot be recovered. For example

$$\mathsf{p} \to \mathsf{h} : \ell \ {}^{\mathsf{h}}\!\bowtie^{\mathsf{k}} \mathsf{k} \to \mathsf{s} : \{\ell, \ell'\} = \mathsf{p} \to \mathsf{s} : \ell$$

but $\textsc{Dec}^{\mathsf{h},\mathsf{k}}(\mathsf{p} \to \mathsf{s} : \ell, \{\mathsf{p}\}, \{\mathsf{s}\}) = (\mathsf{p} \to \mathsf{h} : \ell, \mathsf{k} \to \mathsf{s} : \ell)$.

In the remainder of this section we discuss the decomposition of typed sessions. For that, a notion of process projection is handy. This projection is again parametrised on a set $\mathcal{P}$ of participants and a new participant $\mathsf{h} \notin \mathcal{P}$. The communications of the process with participants in $\mathcal{P}$ remain unchanged. Instead the communications of the process with participants not in $\mathcal{P}$ become communications with h.

Definition 9.6 (Process Projection for Decomposition). *The projection of a process P with respect to a set of participants $\mathcal{P}$ and one interface participant $\mathsf{h} \notin \mathcal{P}$ is the process $P{\Uparrow}^{\mathsf{h}}_{\mathcal{P}}$ defined by:*

$$\mathbf{0}{\Uparrow}^{\mathsf{h}}_{\mathcal{P}} = \mathbf{0} \qquad (\mathsf{p}\dagger\Lambda){\Uparrow}^{\mathsf{h}}_{\mathcal{P}} = \begin{cases} \mathsf{p} \dagger \{\ell.P^{\ell}{\Uparrow}^{\mathsf{h}}_{\mathcal{P}} |\ \ell.P^{\ell} \in \Lambda\} & \text{if } \mathsf{p} \in \mathcal{P}, \\ \mathsf{h} \dagger \{\ell.P^{\ell}{\Uparrow}^{\mathsf{h}}_{\mathcal{P}} |\ \ell.P^{\ell} \in \Lambda\} & \text{if } \mathsf{p} \notin \mathcal{P} \end{cases}$$

where $\dagger$ stands for $?$ or $!$.

It is easy to verify that if $\mathtt{ptp}(P) \subseteq \mathcal{P}$, then $P{\Uparrow}^{\mathsf{h}}_{\mathcal{P}} = P$. Moreover, if $\mathcal{P} = \emptyset$, then $\mathtt{ptp}(P{\Uparrow}^{\mathsf{h}}_{\mathcal{P}}) = \{\mathsf{h}\}$.

We look for a decomposition procedure at the multiparty-session level, through the use of the projection in Definition 9.1. Notice that if $\vdash \mathcal{M} : \mathsf{G}$ and $\{\mathcal{P}, \mathcal{Q}\}$ is a partition of $\mathtt{ptg}(\mathsf{G})$, then $\mathcal{M} \equiv \Pi_{\mathsf{p}\in\mathcal{P}}\mathsf{p} \triangleright P_{\mathsf{p}} \mid \Pi_{\mathsf{q}\in\mathcal{Q}}\mathsf{q} \triangleright Q_{\mathsf{q}}$. We then decompose $\mathcal{M}$ in a session with participants $\mathcal{P}$ plus h and a session with participants $\mathcal{Q}$ plus k. The processes for the participants in $\mathcal{P} \cup \mathcal{Q}$ are obtained by means of the process projection in Definition 9.6 with respect to either $\mathcal{P}$ and h or $\mathcal{Q}$ and k. Instead the processes for h and k are the standard projections of the global types G_1, G_2 obtained by decomposing G with respect to h and k. The typability of the resulting multiparty session is assured by the typing decomposition theorem (Theorem 9.9 below).

Definition 9.7 (Multiparty Session Decomposition). *Let $\vdash \mathcal{M} : \mathsf{G}$ and $\textsc{Dec}^{\mathsf{h},\mathsf{k}}(\mathsf{G}, \mathcal{P}, \mathcal{Q}) = (\mathsf{G}_1, \mathsf{G}_2)$ and*

$$\mathcal{M} \equiv \Pi_{\mathsf{p}\in\mathcal{P}}\mathsf{p} \triangleright P_{\mathsf{p}} \mid \Pi_{\mathsf{q}\in\mathcal{Q}}\mathsf{q} \triangleright Q_q$$

We define:

$$\textsc{Dec}^{\mathsf{h},\mathsf{k}}(\mathcal{M},\mathsf{G},\mathcal{P},\mathcal{Q}) = \big(\Pi_{\mathsf{p}\in\mathcal{P}}\mathsf{p} \triangleright P_{\mathsf{p}}\Uparrow^{\mathsf{h}}_{\mathcal{P}} \mid \mathsf{h} \triangleright \mathsf{G}_1\upharpoonright_{\mathsf{h}}, \Pi_{\mathsf{q}\in\mathcal{Q}}\mathsf{q} \triangleright Q_{\mathsf{q}}\Uparrow^{\mathsf{k}}_{\mathcal{Q}} \mid \mathsf{k} \triangleright \mathsf{G}_2\upharpoonright_{\mathsf{k}}\big)$$

Notably, the decomposition function above exploits the type of the session.

A last proposition on structural preorder between input processes (with a trivial proof) is handy in showing the final theorem.

Proposition 9.8 (Input Process Preorder). *If* $P \leqslant \mathsf{p}?\Lambda$ *and* $P \leqslant \mathsf{p}?\Lambda'$ *and* $msg(\Lambda) \cap msg(\Lambda') = \emptyset$*, then* $P \leqslant \mathsf{p}?(\Lambda \uplus \Lambda')$.

The main result of this section is that the sessions obtained by decomposition can be typed by the global types obtained by decomposition.

Theorem 9.9 (Typing Decomposition). *If* $\vdash \mathcal{M} : \mathsf{G}$ *and* $\textsc{Dec}^{\mathsf{h},\mathsf{k}}(\mathsf{G},\mathcal{P},\mathcal{Q}) = (\mathsf{G}_1,\mathsf{G}_2)$ *and* $\textsc{Dec}^{\mathsf{h},\mathsf{k}}(\mathcal{M},\mathsf{G},\mathcal{P},\mathcal{Q}) = \big(\mathcal{M}_1,\mathcal{M}_2\big)$*, then* $\vdash \mathcal{M}_1 : \mathsf{G}_1$ *and* $\vdash \mathcal{M}_2 : \mathsf{G}_2$.

Proof. It is enough to show that $P \leqslant \mathsf{G}\upharpoonright_{\mathsf{p}}$ implies $P\Uparrow^{\mathsf{h}}_{\mathcal{P}} \leqslant (\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}})\upharpoonright_{\mathsf{p}}$ with $\mathsf{p} \in \mathcal{P}$. The proof is by coinduction on G and by cases on its shape. The case $\mathsf{G} = \mathtt{End}$ is trivial.

If $\mathsf{G} = \mathsf{q} \to \mathsf{p} : \{\ell_i.\mathsf{G}_i \mid {\scriptstyle 1\leq i\leq n}\}$, by definition of projection on a single participant (Definition 3.3) and rule [SUB-IN] for the structural preorder (cf. Definition 3.9) we have

$$P = \mathsf{q}?\{\ell_i.P_i \mid {\scriptstyle 1\leq i\leq m}\} \quad \text{and} \quad \mathsf{G}\upharpoonright_{\mathsf{p}} = \mathsf{q}?\{\ell_i.\mathsf{G}_i\upharpoonright_{\mathsf{p}} \mid {\scriptstyle 1\leq i\leq n}\}$$

with $n \leq m$ and $P_i \leqslant \mathsf{G}_i\upharpoonright_{\mathsf{p}}$ for all i, $1 \leq i \leq n$. By coinduction we get $P_i\Uparrow^{\mathsf{h}}_{\mathcal{P}} \leqslant (\mathsf{G}_i\Uparrow^{\mathsf{h}}_{\mathcal{P}})\upharpoonright_{\mathsf{p}}$ for all i, $1 \leq i \leq n$. By definitions of projections for decomposition (Definitions 9.6 and 9.1)

$$P\Uparrow^{\mathsf{h}}_{\mathcal{P}} = \mathsf{s}?\{\ell_i.P_i\Uparrow^{\mathsf{h}}_{\mathcal{P}} \mid {\scriptstyle 1\leq i\leq m}\} \quad \text{and} \quad (\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}})\upharpoonright_{\mathsf{p}} = \mathsf{s}?\{\ell_i.(\mathsf{G}_i\Uparrow^{\mathsf{h}}_{\mathcal{P}})\upharpoonright_{\mathsf{p}} \mid {\scriptstyle 1\leq i\leq n}\}$$

where $\mathsf{s} = \mathsf{q}$ if $\mathsf{q} \in \mathcal{P}$ and $\mathsf{s} = \mathsf{h}$ otherwise. Then we get $P\Uparrow^{\mathsf{h}}_{\mathcal{P}} \leqslant (\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}})\upharpoonright_{\mathsf{p}}$ by using the rule [SUB-IN] for structural preorder (Definition 3.9).

If $\mathsf{G} = \mathsf{q} \to \mathsf{p} : \Gamma$ the proof is similar to the previous case using the rule [SUB-OUT] of Definition 3.9.

If $\mathsf{G} = \mathsf{r} \to \mathsf{s} : \{\ell_i.\mathsf{G}_i \mid {\scriptstyle 1\leq i\leq n}\}$ and $\mathsf{p} \notin \{\mathsf{r},\mathsf{s}\}$, then we have two cases. If $\{\mathsf{r},\mathsf{s}\} \cap \mathcal{P} = \emptyset$, then by definition of type projection for decomposition

(Definition 9.1), we have $(\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}})\upharpoonright_{\mathsf{p}} = \widehat{\mathsf{G}}\upharpoonright_{\mathsf{p}}$ for some global type $\widehat{\mathsf{G}}$ such that $\widehat{\mathsf{G}} = \mathsf{G}_i\Uparrow^{\mathsf{h}}_{\mathcal{P}}$ for all i, $1 \leq i \leq n$. By the properties of projection for decomposition (Proposition 9.2), $\mathsf{G}_i\upharpoonright_{\mathsf{p}} = \mathsf{G}_1\upharpoonright_{\mathsf{p}}$ for all i, $1 \leq i \leq n$. So coinduction applies. Otherwise, when $\{\mathsf{r},\mathsf{s}\} \cap \mathcal{P} \neq \emptyset$, by Definition 3.3 of projection on a single participant, we have two subcases:

1. $\mathsf{G}\upharpoonright_{\mathsf{p}} = \mathsf{G}_1\upharpoonright_{\mathsf{p}}$ if $\mathsf{G}_i\upharpoonright_{\mathsf{p}} = \mathsf{G}_1\upharpoonright_{\mathsf{p}}$ for all i, $1 \leq i \leq n$;
2. $\mathsf{G}\upharpoonright_{\mathsf{p}} = \mathsf{t}?(\Lambda_1 \uplus \ldots \uplus \Lambda_n)$ with $\mathsf{G}_i\upharpoonright_{\mathsf{p}} = \mathsf{t}?\Lambda_i$ for all i, $1 \leq i \leq n$.

In case 1 the proof is as before. In case 2, from $\Lambda = \Lambda_1 \uplus \ldots \uplus \Lambda_n$ using rule [SUB-IN] of Definition 3.9, we get $\mathsf{G}\upharpoonright_{\mathsf{p}} \leqslant \mathsf{G}_i\upharpoonright_{\mathsf{p}}$, which implies $P \leqslant \mathsf{G}_i\upharpoonright_{\mathsf{p}}$ for all i, $1 \leq i \leq n$. By coinduction $P\Uparrow^{\mathsf{h}}_{\mathcal{P}} \leqslant (\mathsf{G}_i\Uparrow^{\mathsf{h}}_{\mathcal{P}})\upharpoonright_{\mathsf{p}}$ for all i, $1 \leq i \leq n$. By definition of type projection for decomposition (Definition 9.1), we get $(\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}})\upharpoonright_{\mathsf{p}} = \mathsf{t}'?(\Lambda'_1 \uplus \ldots \uplus \Lambda'_n)$ and $(\mathsf{G}_i\Uparrow^{\mathsf{h}}_{\mathcal{P}})\upharpoonright_{\mathsf{p}} = \mathsf{t}'?\Lambda'_i$ (where either $\mathsf{t}' = \mathsf{t}$ or $\mathsf{t}' = \mathsf{h}$) and $msg(\Lambda'_i) = msg(\Lambda_i)$ for all i, $1 \leq i \leq n$. We conclude $P\Uparrow^{\mathsf{h}}_{\mathcal{P}} \leqslant (\mathsf{G}\Uparrow^{\mathsf{h}}_{\mathcal{P}})\upharpoonright_{\mathsf{p}}$ by repeated application of the property of input process preorder (Proposition 9.8). □

Example 9.10. If $\mathsf{G}_0 = \mathsf{p} \to \mathsf{r} : \ell.\mathsf{p} \to \mathsf{s} : \ell.\mathsf{G}_0$ and $\mathcal{P} = \{\mathsf{p}\}$ and $\mathcal{Q} = \{\mathsf{r},\mathsf{s}\}$, then we get $\textsc{Dec}^{\mathsf{h},\mathsf{k}}(\mathsf{G}_0,\mathcal{P},\mathcal{Q}) = \big(\mathsf{G},\mathsf{G}'\big)$ where $\mathsf{G} = \mathsf{p} \to \mathsf{h} : \ell.\mathsf{G}$ and $\mathsf{G}' = \mathsf{k} \to \mathsf{r} : \ell.\mathsf{k} \to \mathsf{s} : \ell.\mathsf{G}'$ as in Example 8.1. Let $\tilde{P} = \mathsf{r}!\ell.\mathsf{s}!\ell.\tilde{P}$, $\tilde{R} = \mathsf{p}?\ell.\tilde{R}$ and $\tilde{S} = \mathsf{p}?\ell.\tilde{S}$ as in Example 8.7. We have

$$\textsc{Dec}^{\mathsf{h},\mathsf{k}}(\mathsf{p} \triangleright \tilde{P} \mid \mathsf{r} \triangleright \tilde{R} \mid \mathsf{s} \triangleright \tilde{S}, \mathsf{G}_0, \mathcal{P}, \mathcal{Q}) = \big(\mathsf{p} \triangleright P \mid \mathsf{h} \triangleright H, \mathsf{r} \triangleright R \mid \mathsf{s} \triangleright S \mid \mathsf{k} \triangleright K\big)$$

where $P = \mathsf{h}!\ell.P$, $H = \mathsf{p}?\ell.H$, $R = \mathsf{k}?\ell.R$, $S = \mathsf{k}?\ell.S$, and $K = \mathsf{r}!\ell.\mathsf{s}!\ell.K$ as in Example 8.1. This shows an instance of Theorem 9.9. ◇

For typed multiparty sessions the decomposition is not the left inverse of the direct composition, as shown by the following example.

Example 9.11. Take

$$\mathcal{M} \equiv \mathsf{p} \triangleright \mathsf{r}!\ell_1 \mid \mathsf{q} \triangleright \mathsf{s}?\ell_2 \mid \mathsf{r} \triangleright \mathsf{p}?\{\ell_1,\ell_3\} \mid \mathsf{s} \triangleright \mathsf{q}!\ell_2$$

and $\mathcal{P} = \{\mathsf{p},\mathsf{q}\}$ and $\mathcal{Q} = \{\mathsf{r},\mathsf{s}\}$ and $\mathsf{G} = \mathsf{p} \to \mathsf{r} : \ell_1.\mathsf{s} \to \mathsf{q} : \ell_2$. We get $\textsc{Dec}^{\mathsf{h},\mathsf{k}}(\mathcal{M},\mathsf{G},\mathcal{P},\mathcal{Q}) = \big(\mathcal{M}_1,\mathcal{M}_2\big)$ where $\mathcal{M}_1 \equiv \mathsf{p}\triangleright\mathsf{h}!\ell_1 \mid \mathsf{q}\triangleright\mathsf{h}?\ell_2 \mid \mathsf{h}\triangleright\mathsf{p}?\ell_1.\mathsf{q}!\ell_2$,

$\mathcal{M}_2 \equiv \mathsf{r} \triangleright \mathsf{k}?\{\ell_1, \ell_3\} \mid \mathsf{s} \triangleright \mathsf{k}!\ell_2 \mid \mathsf{k} \triangleright \mathsf{r}!\ell_1.\mathsf{s}?\ell_2$. Then $\vdash \mathcal{M}_1 : \mathsf{G}_1$ and $\vdash \mathcal{M}_2 : \mathsf{G}_2$ where $\mathsf{G}_1 = \mathsf{p} \to \mathsf{h} : \ell_1.\mathsf{h} \to \mathsf{q} : \ell_2$ and $\mathsf{G}_2 = \mathsf{k} \to \mathsf{r} : \ell_1.\mathsf{s} \to \mathsf{k} : \ell_2$. We obtain

$$\mathcal{M}_1 \,{}^{\mathsf{h}\bowtie\mathsf{k}}\, \mathcal{M}_2 \equiv \mathsf{p} \triangleright \mathsf{r}!\ell_1 \mid \mathsf{q} \triangleright \mathsf{s}?\ell_2 \mid \mathsf{r} \triangleright \mathsf{p}?\ell_1 \mid \mathsf{s} \triangleright \mathsf{q}!\ell_2$$

The difference between $\mathcal{M}$ and $\mathcal{M}_1 \,{}^{\mathsf{h}\bowtie\mathsf{k}}\, \mathcal{M}_2$ is that participant r in $\mathcal{M}$ waits for a message ℓ_3 from participant p. However, p will never send this message. $\diamond$

As shown in this example, decomposing followed by composing typed sessions eliminates useless inputs. This is due to the replacement of some original processes by the projections of global types into the corresponding participants; see the definition of direct composition of multiparty sessions (Definition 8.5).

10. Standard Preorder on Processes

We now discuss the impact of adopting a structural preorder on processes mimicking the standard subtyping relation [22]. We dub *standard preorder* this preorder and we denote it as $\leqslant^+$.

The relation $\leqslant^+$ is obtained by replacing [SUB-OUT] with [SUB-OUT$^+$] below in Definition 3.9, and by replacing the symbol '$\leqslant$'with '$\leqslant^+$'in the other rules.

$$\frac{[\text{SUB-OUT}^+] \qquad P_i \leqslant^+ Q_i \quad \forall 1 \leq i \leq n}{\mathsf{p}!\{\ell_i.P_i \mid 1 \leq i \leq n\} \leqslant^+ \mathsf{p}!(\{\ell_i.Q_i \mid 1 \leq i \leq n\} \uplus \Lambda)} \tag{2}$$

The relation $\leqslant^+$ hence allows more outputs in larger processes than in smaller ones.

Let $\vdash^+$ be the typing system obtained by using $\leqslant^+$ in rule [T-SESS] (cf. Definition 3.10).

Lemmas 4.1 and 4.2 easily adapt to the system $\vdash^+$.

Lemma 10.1 (Inversion Lemma for $\vdash^+$). *If $\vdash^+ \mathcal{M} : \mathsf{G}$ and $\mathsf{p} \triangleright P \in \mathcal{M}$, then $P \leqslant^+ \mathsf{G}\upharpoonright_{\mathsf{p}}$.*

Lemma 10.2 (Canonical Form Lemma for $\vdash^+$). *If $\vdash^+ \mathcal{M} : \mathsf{G}$ and $\mathsf{p} \in \mathtt{ptg}(\mathsf{G})$, then there is $\mathsf{p} \triangleright P \in \mathcal{M}$ and $P \leqslant^+ \mathsf{G}\upharpoonright_{\mathsf{p}}$.*

Clearly $\vdash \mathcal{M} : \mathsf{G}$ implies $\vdash^+ \mathcal{M} : \mathsf{G}$, and a weakening of the vice versa is shown below.

Theorem 10.3 ($\vdash^+$ versus $\vdash$). *If $\vdash^+ \mathcal{M} : \mathsf{G}$, then $\vdash \mathcal{M} : \mathsf{G}'$ for some G'.*

Proof. The proof is by coinduction on G. Let $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \Gamma$. Then, by the canonical form Lemma for $\vdash^+$ (Lemma 10.2) we get $\mathcal{M} \equiv \mathsf{p} \triangleright \mathsf{q}!\Lambda \mid \mathsf{q} \triangleright \mathsf{p}?\Lambda' \mid \mathcal{M}'$ and $\mathsf{q}!\Lambda \leqslant^+ \mathsf{G}\upharpoonright_\mathsf{p}$ and $\mathsf{p}?\Lambda' \leqslant^+ \mathsf{G}\upharpoonright_\mathsf{q}$. By definition of $\leqslant^+$, $msg(\Lambda) \subseteq msg(\Gamma) \subseteq msg(\Lambda')$. Let
$\Lambda = \{\ell_i.P_i \mid {}_{1 \leq i \leq n}\}$, $\Gamma = \{\ell_i.\mathsf{G}_i \mid {}_{1 \leq i \leq n}\} \uplus \Gamma'$ and $\Lambda' = \{\ell_i.Q_i \mid {}_{1 \leq i \leq n}\} \uplus \Lambda''$. By definition of $\leqslant^+$ again, we get $P_i \leqslant^+ \mathsf{G}_i\upharpoonright_\mathsf{p}$ and $Q_i \leqslant^+ \mathsf{G}_i\upharpoonright_\mathsf{q}$ for all i, $1 \leq i \leq n$. Then we can derive $\vdash^+ \mathsf{p} \triangleright P_i \mid \mathsf{q} \triangleright Q_i \mid \mathcal{M}' : \mathsf{G}_i$ for all i, $1 \leq i \leq n$. By coinduction there are G'_i such that $\vdash \mathsf{p} \triangleright P_i \mid \mathsf{q} \triangleright Q_i \mid \mathcal{M}' : \mathsf{G}'_i$ for all i, $1 \leq i \leq n$. We can choose $\mathsf{G}' = \mathsf{p} \to \mathsf{q} : \{\ell_i.\mathsf{G}'_i \mid {}_{1 \leq i \leq n}\}$. □

Thanks to the theorem above, the multiparty sessions that can be typed by $\vdash$ and $\vdash^+$ do coincide.

A first effect of adopting $\leqslant^+$ is a weaker notion of session fidelity (cfr. Theorem 4.9) than the one for $\leqslant$. This is easily illustrated by the following example: using $\vdash^+$, we can type the session $\mathcal{M} = \mathsf{p} \triangleright \mathsf{q}!\ell_1 \mid \mathsf{q} \triangleright \mathsf{p}?\{\ell_1, \ell_2\}$ with $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \{\ell_1, \ell_2\}$, namely $\vdash^+ \mathcal{M} : \mathsf{G}$ holds. However $\mathsf{G} \xrightarrow{\mathsf{p}\ell_i\mathsf{q}} \mathtt{End}$ with $i = 1, 2$, while the only reduction of $\mathcal{M}$ is $\mathcal{M} \xrightarrow{\mathsf{p}\ell_1\mathsf{q}} \mathsf{p} \triangleright \mathbf{0}$. Notice that $\mathsf{G}\upharpoonright_\mathsf{p} = \mathsf{q}!\{\ell_1, \ell_2\}$ and $\mathsf{G}\upharpoonright_\mathsf{q} = \mathsf{p}?\{\ell_1, \ell_2\}$, hence $\mathsf{q}!\ell_1 \leqslant^+ \mathsf{G}\upharpoonright_\mathsf{p}$ but $\mathsf{q}!\ell_1 \not\leqslant \mathsf{G}\upharpoonright_\mathsf{p}$. This is reminiscent of the notion of *whole-spectrum realisation* introduced in [7, 8] and that indeed rules out sessions that bluntly "ignore" some branches in the choices occurring in the global type.

The monotonicity of gateways Lemma (Lemma 5.6) does not hold for $\vdash^+$, as shown by the following counterexample. If $P = \mathsf{p}!\ell_1$ and $Q = \mathsf{p}!\{\ell_1, \ell_2\}$, then $P \leqslant^+ Q$, but $\mathsf{gw}(P, \mathsf{h}) = \mathsf{h}?\ell_1.\mathsf{p}!\ell_1 \not\geqslant^+ \mathsf{h}?\{\ell_1.\mathsf{p}!\ell_1, \ell_2.\mathsf{p}!\ell_1\} = \mathsf{gw}(Q, \mathsf{h})$. The reason is that $\leqslant^+$ allows larger processes to contain additional messages in send operations, but the gateway construction would introduce additional messages in inputs.

Instead we can show the analogous of Lemmas 5.7 and 6.2 for $\leqslant^+$.

Lemma 10.4 (Compatibility is Upward-Closed under $\leqslant^+$). *If $P \leftrightarrow Q$, then $P \leqslant^+ P'$ and $Q \leqslant^+ Q'$ imply $P' \leftrightarrow Q'$.*

Proof. Let us assume

$$\begin{array}{lcll} P = \mathsf{p}!\{\ell_i.P_i \mid {\scriptstyle 1\le i\le n}\} & \leqslant^+ & P' = \mathsf{p}!\{\ell_i.P'_i \mid {\scriptstyle 1\le i\le m}\} \\ \updownarrow & & \\ Q = \mathsf{q}?\{\ell_i.Q_i \mid {\scriptstyle 1\le i\le n'}\} & \leqslant^+ & Q' = \mathsf{q}?\{\ell_i.Q'_i \mid {\scriptstyle 1\le i\le n''}\} \quad \text{with } n'' \le n' \le n \le m \end{array}$$

From $P \leftrightarrow Q$ we get $P_i \leftrightarrow Q_i$ for all i, $1 \le i \le n'$. From $P \leqslant^+ P'$ we get $P_i \leqslant^+ P'_i$ for all i, $1 \le i \le n$. From $Q \leqslant^+ Q'$ we get $Q_i \leqslant^+ Q'_i$ for all i, $1 \le i \le n''$. By coinduction we have $P'_i \leftrightarrow Q'_i$ for all i, $1 \le i \le n''$. We can then conclude $P' \leftrightarrow Q'$. □

Lemma 10.5 (Compatible Sessions have Compatible Types in $\vdash^+$). *If* $(\mathcal{M}, \mathsf{h}) \leftrightarrow (\mathcal{M}', \mathsf{k})$ *and* $\vdash^+ \mathcal{M} : \mathsf{G}$ *and* $\vdash^+ \mathcal{M}' : \mathsf{G}'$*, then*

$$(\mathsf{G}, \mathsf{h}) \leftrightarrow (\mathsf{G}', \mathsf{k}).$$

Proof. The proof mimics that of Lemma 6.2, just using Lemma 10.4 instead of Lemma 5.7. □

A main drawback of rule [SUB-OUT$^+$] is that the typing of gateway composition (cfr. Theorem 6.10) fails to hold due to the fact that $\vdash^+$ breaks the monotonicity of gateways (cfr. Lemma 5.6). We can anyway prove a similar result for $\vdash^+$.

Theorem 10.6 (Typing Gateway Composition in $\vdash^+$). *If* $(\mathcal{M}, \mathsf{h}) \leftrightarrow (\mathcal{M}', \mathsf{k})$ *and* $\vdash^+ \mathcal{M} : \mathsf{G}$ *and* $\vdash^+ \mathcal{M}' : \mathsf{G}'$*, then* $\vdash^+ \mathcal{M} \,{}^{\mathsf{h}\not\leftrightarrow\mathsf{k}}\, \mathcal{M}' : \mathsf{G}''$ *for some* G''*.*

Proof. Theorem 10.3 implies $\vdash \mathcal{M} : \mathsf{G}_1$ and $\vdash \mathcal{M}' : \mathsf{G}_2$ for some $\mathsf{G}_1, \mathsf{G}_2$. Theorem 6.10 gives $\vdash \mathcal{M} \,{}^{\mathsf{h}\not\leftrightarrow\mathsf{k}}\, \mathcal{M}' : \mathsf{G}_1 \,{}^{\mathsf{h}\not\leftrightarrow\mathsf{k}}\, \mathsf{G}_2$. We can choose $\mathsf{G}'' = \mathsf{G}_1 \,{}^{\mathsf{h}\not\leftrightarrow\mathsf{k}}\, \mathsf{G}_2$. This concludes the proof. □

Compatibility of global types is not necessary to get deadlock-freedom of composition via gateways for the type system $\vdash^+$. Let

$$\begin{array}{rclcrcl} \mathsf{G} & = & \mathsf{p} \to \mathsf{h} : \{\ell_1, \ell_2\} & \qquad & \mathsf{G}' & = & \mathsf{k} \to \mathsf{s} : \{\ell_1, \ell_2.\mathsf{s} \to \mathsf{k} : \ell_3\} \\ \mathcal{M} & = & \mathsf{p} \triangleright \mathsf{h}!\ell_1 \mid \mathsf{h} \triangleright \mathsf{p}?\{\ell_1, \ell_2\} & & \mathcal{M}' & = & \mathsf{s} \triangleright \mathsf{k}?\{\ell_1, \ell_2.\mathsf{k}!\ell_3\} \mid \mathsf{k} \triangleright \mathsf{s}!\ell_1 \end{array}$$

Then $\neg\big((\mathsf{G}, \mathsf{h}) \leftrightarrow (\mathsf{G}', \mathsf{k})\big)$ and $\vdash^+ \mathcal{M} : \mathsf{G}$ and $\vdash^+ \mathcal{M}' : \mathsf{G}'$, but

$$\mathcal{M}'' = \mathsf{p} \triangleright \mathsf{h}!\ell_1 \mid \mathsf{s} \triangleright \mathsf{k}?\{\ell_1, \ell_2.\mathsf{k}!\ell_3\} \mid \mathsf{h} \triangleright \mathsf{p}?\{\ell_1.\mathsf{k}!\ell_1, \ell_2.\mathsf{k}!\ell_2\} \mid \mathsf{k} \triangleright \mathsf{h}?\ell_1.\mathsf{s}!\ell_1$$

reduces only as follows:

$$\begin{array}{rcl}\mathcal{M}'' & \xrightarrow{\mathsf{p}\ell_1\mathsf{h}} & \mathsf{s} \triangleright \mathsf{k}?\{\ell_1, \ell_2.\mathsf{k}!\ell_3\} \mid \mathsf{h} \triangleright \mathsf{k}!\ell_1 \mid \mathsf{k} \triangleright \mathsf{h}?\ell_1.\mathsf{s}!\ell_1 \\ & \xrightarrow{\mathsf{h}\ell_1\mathsf{k}} & \mathsf{s} \triangleright \mathsf{k}?\{\ell_1, \ell_2.\mathsf{k}!\ell_3\} \mid \mathsf{k} \triangleright \mathsf{s}!\ell_1 \\ & \xrightarrow{\mathsf{k}\ell_1\mathsf{s}} & \mathsf{s} \triangleright \mathbf{0}\end{array}$$

As this example shows, if the cause of $\neg\big((\mathsf{G}, \mathsf{h}) \leftrightarrow (\mathsf{G}', \mathsf{k})\big)$ occurs after an output that is not present in $\mathcal{M}$ and $\mathcal{M}'$, then no deadlock arises by composing them via gateways.

Instead we can show that the direct composition of multiparty sessions can be typed using the direct composition of global types, as in the system $\vdash$, and differently with respect to the composition via gateways. We define $\mathcal{M} \,{}^{\mathsf{h}^+\bowtie^+\mathsf{k}}\, \mathcal{M}'$ as $\mathcal{M} \,{}^{\mathsf{h}\bowtie\mathsf{k}}\, \mathcal{M}'$ (cf. Definition 8.5), but using $\vdash^+$ instead of $\vdash$.

Theorem 10.7 (Typing Direct Composition in $\vdash^+$). *If $(\mathcal{M}, \mathsf{h}) \leftrightarrow (\mathcal{M}', \mathsf{k})$ and $\vdash^+ \mathcal{M} : \mathsf{G}$ and $\vdash^+ \mathcal{M}' : \mathsf{G}'$, then $\vdash^+ \mathcal{M} \,{}^{\mathsf{h}^+\bowtie^+\mathsf{k}}\, \mathcal{M}' : \mathsf{G} \,{}^{\mathsf{h}\bowtie\mathsf{k}}\, \mathsf{G}'$.*

Proof. The proof is as the proof of Theorem 8.6 using Lemma 10.5 instead of Lemma 6.2. □

The decomposition of global types and multiparty sessions is insensible to the difference between $\leqslant$ and $\leqslant^+$ as well. In particular an analogous of Proposition 9.8 holds. This is not surprising since $\leqslant$ and $\leqslant^+$ only differ for outputs.

Proposition 10.8 (Input Process Standard Preorder). *If $P \leqslant^+ \mathsf{p}?\Lambda$ and $P \leqslant^+ \mathsf{p}?\Lambda'$ and $msg(\Lambda) \cap msg(\Lambda') = \emptyset$, then $P \leqslant^+ \mathsf{p}?(\Lambda \uplus \Lambda')$.*

Theorem 10.9 (Typing Decomposition in $\vdash^+$). *If $\vdash^+ \mathcal{M} : \mathsf{G}$, $\textsc{Dec}^{\mathsf{h},\mathsf{k}}(\mathsf{G}, \mathcal{P}, \mathcal{Q}) = (\mathsf{G}_1, \mathsf{G}_2)$ and $\textsc{Dec}^{\mathsf{h},\mathsf{k}}(\mathcal{M}, \mathsf{G}, \mathcal{P}, \mathcal{Q}) = \big(\mathcal{M}_1, \mathcal{M}_2\big)$, then $\vdash^+ \mathcal{M}_1 : \mathsf{G}_1$ and $\vdash^+ \mathcal{M}_2 : \mathsf{G}_2$.*

Proof. Like in the proof of the typing decomposition Theorem for $\vdash$ (Theorem 9.9), it is enough to show that $P \leqslant^+ \mathsf{G}\upharpoonright_{\mathsf{p}}$ implies $P \Uparrow^{\mathsf{h}}_{\mathcal{P}} \leqslant^+ (\mathsf{G} \Uparrow^{\mathsf{h}}_{\mathcal{P}}) \upharpoonright_{\mathsf{p}}$ with $\mathsf{p} \in \mathcal{P}$. Here we show only the cases whose proofs differ from those considered in Theorem 9.9.

If $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \{\ell_i.\mathsf{G}_i \mid {\scriptstyle 1 \leq i \leq n}\}$, by the definition of projection on a single participant (Definition 3.3) we have

$$P = \mathsf{q}!\{\ell_i.P_i \mid {\scriptstyle 1 \leq i \leq n}\} \quad \text{and} \quad \mathsf{G}{\upharpoonright_\mathsf{p}} = \mathsf{q}!\{\ell_i.\mathsf{G}_i{\upharpoonright_\mathsf{p}} \mid {\scriptstyle 1 \leq i \leq m}\}$$

with $n \leq m$ and $P_i \leqslant^+ \mathsf{G}_i{\upharpoonright_\mathsf{p}}$ for all i, $1 \leq i \leq n$, by [SUB-OUT$^+$] (cf. rule (2) on page 54). By coinduction, $P_i{\Uparrow^\mathsf{h}_\mathcal{P}} \leqslant^+ (\mathsf{G}_i{\Uparrow^\mathsf{h}_\mathcal{P}}){\upharpoonright_\mathsf{p}}$ for all i, $1 \leq i \leq n$. By the definitions of process and type projection for decomposition (Definitions 9.6 and 9.1),

$$P{\Uparrow^\mathsf{h}_\mathcal{P}} = \mathsf{s}!\{\ell_i.P_i{\Uparrow^\mathsf{h}_\mathcal{P}} \mid {\scriptstyle 1 \leq i \leq n}\} \quad \text{and} \quad (\mathsf{G}{\Uparrow^\mathsf{h}_\mathcal{P}}){\upharpoonright_\mathsf{p}} = \mathsf{s}!\{\ell_i.(\mathsf{G}_i{\Uparrow^\mathsf{h}_\mathcal{P}}){\upharpoonright_\mathsf{p}} \mid {\scriptstyle 1 \leq i \leq m}\}$$

where $\mathsf{s} = \mathsf{q}$ if $\mathsf{q} \in \mathcal{P}$ and $\mathsf{s} = \mathsf{h}$ otherwise. Then we get $P{\Uparrow^\mathsf{h}_\mathcal{P}} \leqslant^+ (\mathsf{G}{\Uparrow^\mathsf{h}_\mathcal{P}}){\upharpoonright_\mathsf{p}}$ by using rule [SUB-OUT$^+$].

If $\mathsf{G} = \mathsf{r} \to \mathsf{s} : \{\ell_i.\mathsf{G}_i \mid {\scriptstyle 1 \leq i \leq n}\}$ the proof is as for Theorem 9.9, but using Proposition 10.8 instead of Proposition 9.8. □

We think that this section justifies our choice of structural preorder between processes in building composition by gateways, which is mainly due to the stronger notion of session fidelity it enables and to the fact that it eases the definition of composition via gateways and it makes compatibility a necessary condition for lock-freedom.

11. Concluding Remarks, Related Work, and Future Developments

The distinguishing feature of an *open* system of concurrent components is its capacity of communicating with the "outside", i.e. with some environment of the system. This ability provides means for composing open systems into larger systems (which may still be open). In order to compose systems "safely", it is common practice to rely on interface descriptions.

MPST systems [32, 18, 39] are usually assumed to be *closed*, since all the components needed for the functioning of the system must be already there. As a matter of fact, the MPST framework does work fairly well for the design of closed systems, but does not possess the flexibility open systems can offer. In [1, 2] a novel approach to open systems has been proposed where, according to the current needs, the behaviour of any participant can be regarded as an "interface". An interface is hence intended to represent - somehow dually with respect to the standard notion of interface - part of the expected communication behaviour of the environment. Identifying a participant behaviour as interface corresponds to expecting such a behaviour to be realised

by the environment, rather than by an actual component of the system. Then, according to such an approach, there is actually no distinction between a closed and an open system. In particular, once two systems possess two "compatible" interfaces, they can be composed. The composition mechanism of [1, 2] uses suitable forwarders, dubbed "gateways", for this purpose. The gateways are automatically synthesised out of the compatible interfaces and the composition of two systems simply consists in replacing the latter by the former. It is worth noticing that such a "participants-as-interfaces" approach to compositionality does not affect the expected good features of standard modular techniques for system development, in particular for what concerns the notions of reuse and substitution of code. In fact a system has just to interpret an interface as the "dual"of the standard meaning. As far as we know, modularity of choreographic models has not been explored much. An exception is the approach recently proposed in [14, 42]. Modularity is attained by means of a specified protocol regulating the communications at the "interface". Roughly speaking, this protocol represents the composition interface that rules out, among the communications of components, those not allowed in the composition. Components in [14, 42] are *reactive* (similarly to [24, 25]). More precisely, components exports their *input* and *output* ports which are connected by links. A component is a computational unit that applies local transformations on data arriving at their input ports in order to produce events at their output ports. The typing discipline in [42] is not concerned with the preservation of well-formedness of the system; rather, it tracks down data flow dependencies across the communications specified in the protocol.

In the present paper we have provided a MPST formalism where global types are used to describe the overall behaviour of "safe" systems of processes – dubbed *multiparty sessions* – directly related to the global types projections. We have then adapted and extended to such a formalism the approach of [1, 2], both at the global type and multiparty session levels. In general, managing to look at global types as overall descriptions of open systems results in a MPST-based approach to the modular design of systems. From another point of view, by means of such an approach, one could develop systems where some participants, instead of representing actual processes, describe API calls, along the line of what some researchers refer to as Behavioural API [6]. Moreover, this approach is helpful also after the system implementation phase. Let us assume to have a system developed using a MPST software development approach. After the implementation phase, one

could realise that the service corresponding to a participant of the system can be more suitably provided by another system. The participant can then be safely replaced by a gateway composition with the other system, and the composition operation on global types enables a global view of what is going on in the resulting system.

Our calculus of multiparty sessions is like those of [23, 29], but for the use of coinduction instead of induction, which is inspired by [16, 43]. As in [43] we get rid of local types, which in many calculi are similar to processes [17, 23, 29]. The syntax of global types is the coinductive version of the standard syntax [32]. With respect to [43], we relax the conditions imposed on global types in order to be projectable, in particular ensuring projectability of global types resulting from our composition operation (a property that does not hold in the formalism of [43]).

A relevant feature of our formalism is that the composition by gateways operation on systems can be lifted to the level of global types. In [1, 2], where systems of CFSMs were considered, such a lifting was done by extending the syntax of global descriptions with a *new* symbol, whose semantics is indeed the composition by gateways at system level. Instead, in the present paper, the session obtained by composition is represented using the standard syntax of global types. Moreover, we have shown that the compatibility relation of [1, 2], which requires duality, can be relaxed to a relation very close to session types subtyping [28, 22]. We further investigated the notion of compatibility by also showing that compatibility of global types is actually also a necessary condition in order to get a "safe" composition of systems.

Our formalism is equipped with a structural preorder on processes akin to the subtyping relation between session types of [17], which in turn is a restriction of the subtyping of [22]. This choice is justified by the fact that the subtyping of [22] allows process substitution, while the subtyping of [28] allows channel substitution, as observed in [27].

We have also shown that the composition by gateways technique can be extended so that the gateways can be actually dropped. This alternative approach has led to the definition of another composition operation that we dubbed direct. The use of direct composition - in particular at type level - yields a powerful tool. In fact, it enables the modular design and development of distributed systems, expecially in case such an operation can be paired – as we did – with another one enabling a global description to be decomposed. Small systems can be separately implemented out of the types obtained by decomposing a global type. Then by direct composition such systems can be

merged into a system implementing the original global description, as shown by our results (some constraints have to be imposed on the global type to be decomposed).

A final contribution of the present paper is the discussion on how the properties of the given constructions change when we consider a structural preorder on processes mimicking the subtyping relation of [22].

In [37, 36] global types are built out of several local types (under certain conditions). We also aim at getting global types, but we obtain them from the global types describing the two systems which are composed.

The dynamic addition/removal of participants (they can join/leave the session after it has been set up) is supported in the calculus of [33] and recently in [11]. In [33] the extension of a system is part of the global protocol. The extension operation is sort of internalised. We take instead the standard point of view of open systems, where the possible extensions cannot be decided in advance. This is also the point of view adopted in [11], however the type system proposed in [11] aims to guarantee data-flow properties instead of control properties. For instance, in [11] well-typed systems may have locks. Both those approaches to the dynamic system extension issue look orthogonal to the work presented here.

Another approach to safe composition of systems allows one to replace a fragment of a choreography (essentially a program structured as a multiparty session type) with a new one at run-time [21], possibly also introducing new participants [26]. However, the new fragment could not communicate by exchanging messages with the context choreography, but only through shared state. Also, the change may deeply impact all the participants, and a complex middleware managing the change is needed. Hence, this approach is orthogonal to ours as well.

Both arbiter processes [13] and mediums [12] coordinate communications described by global types. A difference with the present paper is that their aim is to reduce the interactions in multiparty sessions to interactions in binary sessions. Our gateways do instead act as simple forwarders, with the aim of composing two multiparty systems. Nonetheless, our work could be further developed in the logical context of [13]: in the logical interpretation of multiparty sessions one could introduce a "composition cut" corresponding to a sort of composition-by-gateways operator. Then the good properties of the system corresponding to the proof containing the cut should be guaranteed by proving that the "composition cut" is actually an admissible rule. The proof should consist of a "composition cut elimination" procedure corresponding to

our direct composition on global types, which bypasses the use of gateways.

The results of this paper would be more applicable accounting for asynchronous communications. In particular, a first relevant step would consist in allowing gateways to interact asynchronously. We expect that compatibility could be extended, since the subtyping for asynchronous multiparty sessions is more permissive than the subtyping for the synchronous ones [39]. Of course this extension requires care, since the subtyping of [39] is undecidable, as shown in [10, 38].

The composition via gateways proposed in [1, 2] and exploited in the present paper in a multiparty session setting does produce, by repeated composition, networks of systems possessing a tree-like topology. More precisely, after having composed two global types on a pair of interfaces h and k, one can take the result and compose it with a further global type, on another pair of interfaces h_1 and k_1. One can iterate this procedure, and the graph where global types are nodes and connections are edges of the resulting system is actually a tree. In order to get graph topologies allowing for cycles, it sounds natural to extend the present single interface composition to a multiple interfaces one, or to allow connections among two roles of the same global type. Such an extension, however, immediately reveals itself to be unsound: by composing via gateways more than one pair of compatible interfaces one could obtain a deadlocked system. A very simple example for that is

$$\mathsf{G} = \mathsf{p} \to \mathsf{h} : \ell \quad \text{and} \quad \mathsf{G}' = \mathsf{k} \to \mathsf{s} : \ell$$

By projection we get the systems

$$\mathcal{M} = \mathsf{p} \triangleright \mathsf{h}!\ell \mid \mathsf{h} \triangleright \mathsf{p}?\ell \quad \text{and} \quad \mathcal{M}' = \mathsf{k} \triangleright \mathsf{s}!\ell \mid \mathsf{s} \triangleright \mathsf{k}?\ell$$

It is immediate to check that p and s are compatible, as well as h and k. Simultaneously connecting $\mathcal{M}$ and $\mathcal{M}'$ through both the compatible pairs (p, s) and (h, k) would result in the following deadlocked system

$$\mathsf{p} \triangleright \mathsf{s}?\ell.\mathsf{h}!\ell \quad | \quad \mathsf{h} \triangleright \mathsf{p}?\ell.\mathsf{k}!\ell \quad | \quad \mathsf{k} \triangleright \mathsf{h}?\ell.\mathsf{s}!\ell \quad | \quad \mathsf{s} \triangleright \mathsf{k}?\ell.\mathsf{p}!\ell$$

(This example is similar to one developed in the CFSMs setting of [2]).
It is easy to realise that this sort of difficulty stays the same also in case direct composition is considered. In order to guarantee "safeness" of multiple connections, suitable requirements have hence to be devised. An adaptation to the present setting of the *interaction type system* of [19] could be investigated in future for such an aim.

Acknowledgments. We gratefully acknowledge the fruitful interactions and communications with the anonymous referees through the ICE Forum.

The present version of this paper strongly improved with respect to the journal submission thanks to the careful reports. The referees did a great job in pointing out many places where the technical details appeared without the needed explanations. The difference between the two versions are several illustrating discussions with enlightening examples.

References

[1] Barbanera, F., de'Liguoro, U., Hennicker, R., 2018. Global Types for Open Systems, in: ICE, Open Publishing Association. pp. 4–20.

[2] Barbanera, F., de'Liguoro, U., Hennicker, R., 2019. Connecting Open Systems of Communicating Finite State Machines. Journal of Logical and Algebraic Methods in Programming 109, 1–34.

[3] Barbanera, F., Dezani-Ciancaglini, M., 2019. Open Multiparty Sessions, in: ICE, Open Publishing Association. pp. 77–96.

[4] Barbanera, F., Dezani-Ciancaglini, M., de'Liguoro, U., 2014. Compliance for Reversible Client/Server Interactions, in: BEAT, Open Publishing Association. pp. 35–42.

[5] Barbanera, F., Dezani-Ciancaglini, M., de'Liguoro, U., 2016. Reversible Client/Server Interactions. Formal Aspects of Computing 28, 697–722.

[6] BEHAPI website, 2019. Behapi website. `https://www.um.edu.mt/projects/behapi/`.

[7] Bocchi, L., Melgratti, H.C., Tuosto, E., 2014. Resolving Nondeterminism in Choreographies, in: ESOP, Springer. pp. 493–512.

[8] Bocchi, L., Melgratti, H.C., Tuosto, E., 2020. On Resolving Nondeterminism in Choreographies. Logical Methods in Computer Science To appear.

[9] Brand, D., Zafiropulo, P., 1983. On Communicating Finite-State Machines. Journal of the ACM 30, 323–342.

[10] Bravetti, M., Carbone, M., Zavattaro, G., 2017. Undecidability of Asynchronous Session Subtyping. Information and Computation 256, 300–320.

[11] Bruni, R., Corradini, A., Gadducci, F., Melgratti, H.C., Montanari, U., Tuosto, E., 2019. Data-Driven Choreographies à la Klaim, in: Models, Languages, and Tools for Concurrent and Distributed Programming - Essays Dedicated to Rocco De Nicola on the Occasion of His 65th Birthday, Springer. pp. 170–190.

[12] Caires, L., Pérez, J.A., 2016. Multiparty Session Types Within a Canonical Binary Theory, and Beyond, in: FORTE, Springer. pp. 74–95.

[13] Carbone, M., Lindley, S., Montesi, F., Schürmann, C., Wadler, P., 2016. Coherence Generalises Duality: A Logical Explanation of Multiparty Session Types, in: CONCUR, Schloss Dagstuhl. pp. 33:1–33:15.

[14] Carbone, M., Montesi, F., Vieira, H.T., 2018. Choreographies for Reactive Programming. CoRR abs/1801.08107, 1–25.

[15] Cardone, F., Coppo, M., 2013. Recursive Types, in: Barendregt, H., Dekkers, W., Statman, R. (Eds.), Lambda Calculus with Types. Cambridge University Press. Perspectives in Logic, pp. 377–576.

[16] Castagna, G., Gesbert, N., Padovani, L., 2009. A Theory of Contracts for Web Services. ACM Transactions on Programming Languages and Systems 31, 19:1–19:61.

[17] Castellani, I., Dezani-Ciancaglini, M., Giannini, P., 2019. Reversible Sessions with Flexible Choices. Acta Informatica 56, 553–583.

[18] Coppo, M., Dezani-Ciancaglini, M., Padovani, L., Yoshida, N., 2015. A Gentle Introduction to Multiparty Asynchronous Session Types, in: Formal Methods for Multicore Programming, Springer. pp. 146–178.

[19] Coppo, M., Dezani-Ciancaglini, M., Yoshida, N., Padovani, L., 2016. Global Progress for Dynamically Interleaved Multiparty Sessions. Mathematical Structures in Computer Science 26, 238–302.

[20] Courcelle, B., 1983. Fundamental Properties of Infinite Trees. Theoretical Computer Science 25, 95–169.

[21] Dalla Preda, M., Gabbrielli, M., Giallorenzo, S., Lanese, I., Mauro, J., 2017. Dynamic Choreographies: Theory and Implementation. Logical Methods in Computer Science 13, 1–57.

[22] Demangeon, R., Honda, K., 2011. Full Abstraction in a Subtyped Pi-Calculus with Linear Types, in: CONCUR, Springer. pp. 280–296.

[23] Dezani-Ciancaglini, M., Ghilezan, S., Jaksic, S., Pantovic, J., Yoshida, N., 2015. Precise Subtyping for Synchronous Multiparty Sessions, in: PLACES, Open Publishing Association. pp. 29–43.

[24] Ferrari, G., Guanciale, R., Strollo, D., Tuosto, E., 2007. Coordination Via Types in an Event-Based Framework, in: FORTE, Springer. pp. 66–80.

[25] Ferrari, G., Strollo, D., Guanciale, R., 2006. JSCL: A Middleware for Service Coordination, in: Formal Methods for Networked and Distributed Systems, Springer. pp. 46–60.

[26] Gabbrielli, M., Giallorenzo, S., Lanese, I., Mauro, J., 2019. Guess Who's Coming: Runtime Inclusion of Participants in Choreographies, in: The Art of Modelling Computational Systems: A Journey from Logic and Concurrency to Security and Privacy - Essays Dedicated to Catuscia Palamidessi on the Occasion of Her 60th Birthday, Springer. pp. 118–138.

[27] Gay, S., 2016. Subtyping Supports Safe Session Substitution, in: A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday, Springer. pp. 95–108.

[28] Gay, S., Hole, M., 2005. Subtyping for Session Types in the Pi Calculus. Acta Informatica 42, 191–225.

[29] Ghilezan, S., Jaksic, S., Pantovic, J., Scalas, A., Yoshida, N., 2019. Precise Subtyping for Synchronous Multiparty Sessions. Journal of Logic and Algebraic Methods in Programming 104, 127–173.

[30] Honda, K., Vasconcelos, V.T., Kubo, M., 1998. Language Primitives and Type Discipline for Structured Communication-Based Programming, in: ESOP, Springer. pp. 122–138.

[31] Honda, K., Yoshida, N., Carbone, M., 2008. Multiparty Asynchronous Session Types, in: POPL, ACM Press. pp. 273–284.

[32] Honda, K., Yoshida, N., Carbone, M., 2016. Multiparty Asynchronous Session Types. Journal of the ACM 63, 9:1–9:67.

[33] Hu, R., Yoshida, N., 2017. Explicit Connection Actions in Multiparty Session Types, in: FASE, Springer. pp. 116–133.

[34] Kobayashi, N., 2002. A Type System for Lock-Free Processes. Information and Computation 177, 122–159.

[35] Kozen, D., Silva, A., 2017. Practical Coinduction. Mathematical Structures in Computer Science 27, 1132–1152.

[36] Lange, J., 2014. On the Synthesis of Choreographies. Ph.D. thesis. Department of Computer Science, University of Leicester.

[37] Lange, J., Tuosto, E., 2012. Synthesising Choreographies from Local Session Types, in: CONCUR, Springer. pp. 225–239.

[38] Lange, J., Yoshida, N., 2017. On the Undecidability of Asynchronous Session Subtyping, in: FOSSACS, Springer. pp. 441–457.

[39] Mostrous, D., Yoshida, N., Honda, K., 2009. Global Principal Typing in Partially Commutative Asynchronous Sessions, in: ESOP, Springer. pp. 316–332.

[40] Padovani, L., 2014. Deadlock and Lock Freedom in the Linear π-calculus, in: LICS, ACM Press. pp. 72:1–72:10.

[41] Pierce, B.C., 2002. Types and Programming Languages. MIT Press.

[42] Savanovic, Z., Vieira, H., Galletta, L., 2020. A Type Language for Message Passing Component-based Systems, in: ICE, Open Publishing Association. To appear.

[43] Severi, P., Dezani-Ciancaglini, M., 2019. Observational Equivalence for Multiparty Sessions. Fundamenta Informaticae 167, 267–305.