Monitoring Networks through Multiparty Session Types ☆

Laura Bocchi[a], Tzu-chun Chen[b], Romain Demangeon[c], Kohei Honda[d], Nobuko Yoshida[e]

[a]*School of Computing, University of Kent, United Kingdom*
[b]*Department of Computer Science, TU Darmstadt, Germany*
[c]*LIP6, Université Pierre et Marie Curie, France*
[d]*Department of Computer Science, Queen Mary, University of London, United Kingdom*
[e]*Department of Computing, Imperial College London, United Kingdom*

Abstract

In large-scale distributed infrastructures, applications are realised through communications among distributed components. The need for methods for assuring safe interactions in such environments is recognised, however the existing frameworks, relying on centralised verification or restricted specification methods, have limited applicability. This paper proposes a new theory of *monitored* π-calculus with dynamic usage of *multiparty session types* (MPST), offering a rigorous foundation for safety assurance of distributed components which asynchronously communicate through multiparty sessions. Our theory establishes a framework for semantically precise decentralised run-time enforcement and provides reasoning principles over monitored distributed applications, which complement existing static analysis techniques. We introduce asynchrony through the means of explicit routers and global queues, and propose novel equivalences between networks, that capture the notion of interface equivalence, i.e. equating networks offering the same services to a user. We illustrate our static-dynamic analysis system with an ATM protocol as a running example and justify our theory with results: satisfaction equivalence, local/global safety and transparency, and session fidelity.

Keywords: Session Types; the Pi - Calculus; Dynamic Monitoring; Runtime Verification; Bisimulation

☆This work has been partially sponsored by Ocean Observatories Initiative and EPSRC EP/K011715/1, EP/G015635/1, EP/G015481/1, EP/K034413/1, EP/L00058X/1 and EP/N027833/1 and EU project FP7-612985 UpScale and COST Action IC1201 BETTY and ERC grant FP7-617805 LiVeSoft Lightweight Verification of Software.

1. Introduction

One of the main challenges in the engineering of distributed systems is the comprehensive verification of distributed software without relying on ad-hoc and expensive testing techniques. Multiparty session types (MPST) is a typing discipline for communication programming, that was originally developed in the π-calculus [33, 6, 9, 23, 24, 16] towards tackling this challenge. The idea is that applications are built starting from units of design called sessions. Each type of session, involving multiple roles, is first modelled from a global perspective (*global type*) and then projected onto *local types*, one for each role involved. As a verification method, the existing MPST systems focus on static type checking of endpoint processes against local types. The standard properties enjoyed by well-typed processes are communication safety (all processes conform to globally agreed communication protocols) and freedom from deadlocks.

The direct application of the theoretical MPST techniques to the current practice, however, presents a few obstacles. First, the existing type systems are targeted at calculi with first class primitives for linear communication channels and communication-oriented control flow; the majority of mainstream engineering languages would need to be extended in this sense to be suitable for syntactic type checking using session types. Unfortunately, it is not always straightforward to add these features to the specific host languages. Furthermore, the executable processes in a distributed system may be implemented in different languages. Second, for domains where dynamically typed or untyped languages are popular (e.g., Web programming), or in multi-organisational scenarios, the introduction of static typing infrastructure to support MPST may not be realistic.

Development of Heterogeneous Systems based on MPSTs. This article proposes a theoretical framework addressing the issues discussed above, by supporting the combination of static *and* dynamic verification of processes communicating in a network. Fig. 1 illustrates the proposed framework. As standard in MPST [33, 6], the first stage is to specify a global protocol as a *global type*, describing how the participants should interact in a multiparty session. The global type is then mechanically *projected* to generate local protocols, as *local types*, specifying the communication behaviour expected of each role in the session. The global type in Fig. 1 involves three roles, yielding three local types upon projection. Next, each principal in a network implements one (or possibly more) local types. We call these implementations endpoint processes, or simply processes. We aim to capture the decentralised nature of distributed application development, providing support for heterogeneous distributed systems by allowing components to be independently implemented, using different languages, libraries and programming techniques. Assume that (1) the process on the right-hand side of

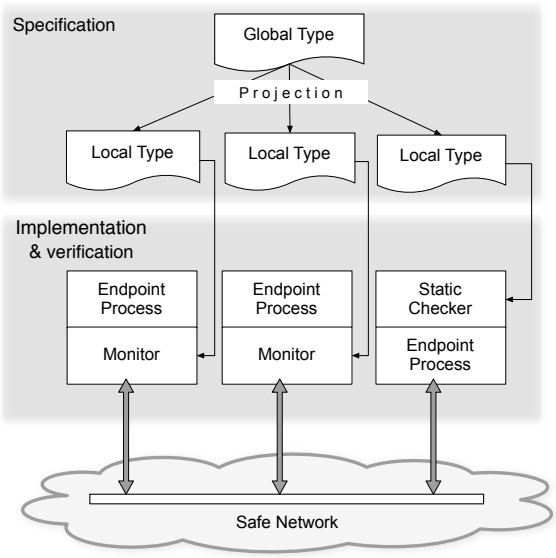

Figure 1: Static/dynamic verification through global types and projection

Fig. 1 is implemented in a language that supports static verification with session typing techniques, and that conformance to the implemented local type is verified this way, and (2) the other two processes are implemented in standard Java and Python, respectively, using simple session programming APIs and are not amenable to static typing. To ensure that the composition of these three processes conforms to the intended protocol we wrap the processes that cannot be statically verified with dedicated distributed *monitors*, that dynamically verify their participation in the session. In other words, our framework allows processes to be independently verified, either statically during deployment, or dynamically during execution, while retaining the strong global safety properties of statically verified systems.

This work is motivated in part by our ongoing collaboration with the Ocean Observatories Initiative (OOI) [44], a project to establish cyberinfrastructure for the delivery, management and analysis of scientific data from a large network of ocean sensor systems. Their architecture relies on the combination of high-level protocol specifications (to express how the infrastructure services should be used) and distributed run-time monitoring to regulate the behaviour of third-party applications in the system. An implementation of the framework in Fig. 1 is currently integrated into the OOI infrastructure. In this implementation, processes are specified using Scribble [47, 51, 31, 30] (a practical incarnation of MPST) and processes are implemented in the Python programming language and dynamically monitored [43, 34].

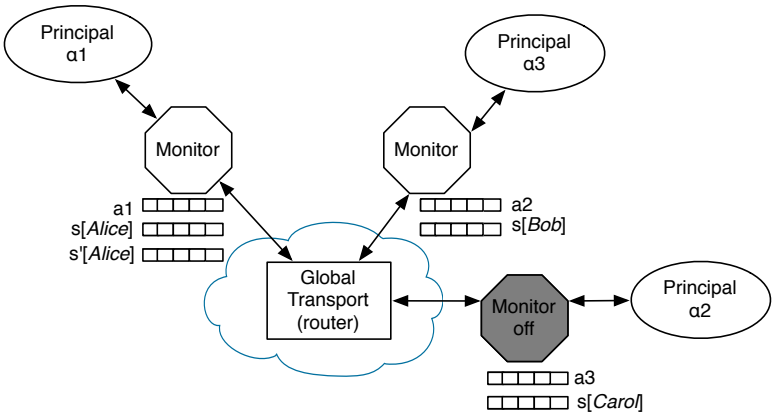

Figure 2: Architecture of monitored/unmonitored networks

Monitored Networks. Networks are organised as follows: a group of principals run processes communicating via asynchronous message passing; dedicated trusted monitors (one for each principal) guard the run-time behaviour of both the environment and that principal, through the evaluation of incoming and outgoing messages. The aim is to protect the principal from violations by other principals, and also to prevent the principal from committing violation (this can be used e.g. for debugging). Monitors regulate (1) the initiation, by principals, of new sessions, each specified by a well-defined global type, and (2) the movement of messages within each session. Fig. 2 illustrates the architecture of a network with three principals ($\alpha 1$, $\alpha 2$ and $\alpha 3$); all principals are monitored except $\alpha 3$, namely we assume the processes run by $\alpha 3$ have been statically checked hence its monitor can be switched-off (indeed all outgoing and incoming messages can pass through without dynamic checking); each principal is associated with one or more *shared queues*, on which all other principals can send invitations to join new sessions. The messages exchanged within a session are all associated to *one common session ID*, and the exchange of messages in a session is regulated by verifying that the causality of messages follows the specification (roughly, the ensemble of local types) of that session. In Fig. 2 each principal is associated with exactly one shared queue, and we denote with ai the queue associated with αi, with $i \in \{1, 2, 3\}$; principal $\alpha 1$ is currently playing role *Alice* in two sessions with session IDs s and s', whereas $\alpha 2$ and $\alpha 3$ are playing *Bob* and *Carol*, respectively, in just one session s (e.g., the invitations to join s' have not yet been received by them).

A Formal Theory for Dynamic Verification. Our theory is based on the idea that, if the endpoint processes in a system are *independently* verified (either statically or dynamically) to conform a local type, then the corresponding global pro-

tocol is respected *as a whole*. To this goal, we propose a new formal model and a bisimulation theory for heterogeneous networks of monitored and unmonitored processes.

For the first time, we model dynamic verification based on types for the π-calculus. We provide an explicit account of the *routing mechanism* that is implicitly present inside the MPST framework: in a session, messages are sent to abstract roles (e.g. to a Seller), and a router (a dynamically updated component of the network) translates these roles into actual addresses.

Our approach also aims at giving a semantical equivalence for a collection of protocols (and networks), by reaching a formal criterion for equating services. By taking the routing feature into account when designing novel equivalences, our formal model can relate networks built in different ways (through different distributions or relocations of services) but offering the same *interface* to an external observer. The router, being in charge of associating roles with principals, hides to an external user the internal composition of a network: what distinguishes two networks is not their structure but the services they are able to provide, or more precisely, the local types they offer to the outside. We prove that bisimulation is compositional (Proposition 4.4) and that equivalent networks satisfy the same specification (Proposition 4.6).

We formally define a satisfaction relation to express when the behaviour of a network conforms to a global specification and we prove a number of properties of our model: *local safety* (Theorem 5.2) states that a monitored process respects its local protocol, i.e. that dynamic verification by monitoring is sound; *global safety* (Theorem 5.4) extends local safety to networks involving multiple principals; *local transparency* (Theorem 6.1) states that a monitored process has equivalent behaviour to an unmonitored but well-behaved (e.g. statically typed) process; and *global transparency* (Theorem 6.3) states that a network where each principal is monitored has equivalent behaviour to an unmonitored but well-behaved network.

Finally, we introduce a stronger property than global safety, *session fidelity* (Theorem 7.13), which not only guarantees conformance of each monitored process in a network to the ensemble of local specifications, but also requires that the overall flow of messages throughout the router is correct. In this way, session fidelity shows the correspondency between the behaviour of a monitored system and the behaviour specified by a global protocol. Together, these properties justify our framework for decentralised verification by allowing monitored and unmonitored processes to be safely mixed while preserving protocol conformance for the entire network. Technically, these properties also ensure the coherence of our theory, by relating the satisfaction relations with the semantics and static validation procedures.

Our theory is more involved than most of the existing works in the domain of session verification [33] as, for the first time, both networks and monitoring

are made explicit. Our abstract model for session networks describe the evolution of the network at a lower-level; for instance, we introduce dynamic update of routing information: a participant taking part in a session does not send a message to another participant, but sends a message to a role which is then routed by the networks to the corresponding participant.

Contributions and Outline. This work is an extended version of [7] that includes: the definitions omitted in [7], additional examples, and full proofs. Specifically, we extended [7] by including the following additional material:

- the formal definition of monitorability, a consistency condition on global types, together with a discussion on its relevance and a statement of its decidability (§ 2.2);
- the detailed definitions, full formal statement and proofs of session fidelity and its relationship with global safety (in this introduction, § 5 and § 7), which is only outlined in [7];
- a simpler but less restrictive semantics of networks (e.g., a principal is now allowed to engage as different participants in the same session);
- a detailed formalisation for behavioural equivalences (§ 4.3)
- a formal statement on global safety in mixed (i.e., monitored *and* unmonitored) networks (Corollary 6.4);

§ 2 and § 3 introduce the formalisms for protocol specifications and networks, respectively. § 3 provides a formal framework for monitored networks based on π-calculus processes and protocol-based run-time enforcement through monitors. § 4 introduces: a semantics for specifications (§ 4.1), a novel behavioural theory for compositional reasoning over monitored networks through the use of equivalences (bisimilarity and barbed congruence) and the satisfaction relation (§ 4.2). Local and global safety are stated and proved in § 5, transparency in § 6, and session fidelity in § 7. Related works are discussed in § 8 and future works in § 9.

2. Monitorability in Multiparty Session Types

This section provides basic definitions and well-formedness conditions for multiparty session types. In § 2.1 we summarise the syntax of multiparty session types annotated with logical assertions (MPST), which we use to model protocols. In § 2.2, we introduce a condition called *monitorability*, enforceable on MPST, that sets the basis for the results presented in the next sections.

$$
\begin{array}{lcl}
A & ::= & \mathsf{tt} \mid \mathsf{ff} \mid e_1 = e_2 \mid e_1 < e_2 \mid \neg A \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \\
e & ::= & v \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 \,\mathtt{mod}\, e_2 \\
S & ::= & \mathsf{bool} \mid \mathsf{int} \mid \mathsf{string} \\
G & ::= & \mathtt{r}_1 \to \mathtt{r}_2 : \{l_i(x_i : S_i)\{A_i\}.G_i\}_{i \in I} \mid G_1 \mid G_2 \mid \mu \mathsf{t}.G \mid \mathsf{t} \mid \mathsf{end} \\
T & ::= & \mathtt{r}!\{l_i(x_i : S_i)\{A_i\}.T_i\}_{i \in I} \mid \mathtt{r}?\{l_i(x_i : S_i)\{A_i\}.T_i\}_{i \in I} \mid \mu \mathsf{t}.T \mid \mathsf{t} \mid \mathsf{end}
\end{array}
$$

Figure 3: Global and local types with assertions

2.1. Multiparty Session Types with Assertions

Multiparty session types with assertions [9] are abstract descriptions of the structure of interactions among the roles in a multiparty session (i.e., in a protocol); they specify the potential flows of messages, the conditions under which interactions may occur, and the constraints on the communicated values.

Global types with assertions, or just *global types*, describe multiparty sessions from a network perspective. Global types can be projected onto local types with assertions, or just *local types*, each describing the protocol from the perspective of a single role.

The syntax of global types ($G, G', \ldots$) and local types ($T, T', \ldots$) is defined in Fig. 3. We let values $v, v', \ldots$ range over boolean constants, numerals and strings, and $e, e', \ldots$ range over first-order expressions. *Assertions*, ranged over by $A, A', \ldots$ are logical predicates used to express constraints on the values communicated. We consider assertions following the grammar given in Fig. 3 although other decidable logics could be used. For instance, in [9, 24] the logics includes existential quantifiers which we have omitted for simplicity (of evaluation of the assertions by the run-time monitors), and because they are not necessary for our run-time theory. The *sorts* of exchanged values ($S, S', \ldots$) consists of atomic types.

Global Types with Assertions. $\mathtt{r}_1 \to \mathtt{r}_2 : \{l_i(x_i : S_i)\{A_i\}.G_i\}_{i \in I}$ models an interaction where role $\mathtt{r}_1$ sends role $\mathtt{r}_2$ one of the *branch labels* l_i, as well as a payload denoted by an *interaction variable* x_i of sort S_i. Interaction variable x_i binds its occurrences in A_i and G_i. A_i is the assertion which needs to hold for $\mathtt{r}_1$ to select l_i, and which may constrain the values instantiating x_i. $G_1 \mid G_2$ specifies two (independent) parallel threads in a session. We assume $G \mid \mathsf{end}$ and $\mathsf{end} \mid G$ are identical with G. $\mu \mathsf{t}.G$ is a recursive type, where G is guarded in the standard way [45, 6], and end ends the session.

*Example 2.1 (*ATM*: the global type).* Global type $G_{\mathtt{ATM}}$ specifies an *ATM* scenario. Each session of *ATM* involves three roles: a client `C`, a payment server `S` and a

separate authenticator A.

$$
\begin{array}{lll}
G_{\mathtt{ATM}} = & \mathtt{C} \to \mathtt{A} : \{ & \mathtt{Login}(x_i : \mathsf{string})\{\mathsf{tt}\}. \\
& \mathtt{A} \to \mathtt{S} : \{ & \mathtt{LoginOK}()\{\mathsf{tt}\}.\ \mathtt{A} \to \mathtt{C} : \{\mathtt{LoginOK}()\{\mathsf{tt}\}.\ G_{\mathtt{LOOP}}\}, \\
& & \mathtt{LoginFail}()\{\mathsf{tt}\}.\ \mathtt{A} \to \mathtt{C} : \{\mathtt{LoginFail}()\{\mathsf{tt}\}.\ \mathsf{end}\}\}\} \\
G_{\mathtt{LOOP}} = & \mu\ \mathtt{LOOP}. & \\
& \mathtt{S} \to \mathtt{C} : \{ & \mathtt{Account}(x_b : \mathsf{int})\{x_b \geq 0\}. \\
& \mathtt{C} \to \mathtt{S} : \{ & \mathtt{Withdraw}(x_p : \mathsf{int})\{x_p > 0 \wedge x_b - x_p \geq 0\}.\ \mathtt{LOOP}, \\
& & \mathtt{Deposit}(x_d : \mathsf{int})\{x_d > 0\}.\ \mathtt{LOOP}, \\
& & \mathtt{Quit}()\{\mathsf{tt}\}.\mathsf{end}\}\}
\end{array}
$$

At the beginning of the session C sends A payload x_i (i.e., the login details); then A decides whether the authentication is successful or not, and informs S and C of the choice by sending either label LoginOK or LoginFail. If LoginFail is chosen then the session terminates. If LoginOK is chosen then C and S enter a loop specified by $G_{\mathtt{Loop}}$. In each iteration of $G_{\mathtt{Loop}}$, S sends C the amount x_b currently available in the account. The predicate states that x_b must be non negative. C can then choose one of the following three labels: Withdraw (withdraws an amount x_p, which must be positive and not exceed the current amount x_b), Deposit (deposits a positive amount x_d in the account), or Quit (ends the session). If either Withdraw or Deposit was chosen then another iteration is executed.

Local Types with Assertions. Each local type T is associated with a role taking part in a session. Local type $\mathtt{r}!\{l_i(x_i : S_i)\{A_i\}.T_i\}_{i \in I}$ models an interaction where the role under consideration (say p) sends r a branch label l_i and a message denoted by an interaction variable x_i of sort S_i. Its dual is the receive interaction $\mathtt{p}?\{l_i(x_i : S_i)\{A_i\}.T_i\}_{i \in I}$, where the role under consideration (say r) receives a message from p. As customary for MPST, only global types can be composed in parallel, namely there is no parallel composition of local types. This is guaranteed by a well-formedness condition on global types (see Definition 2.4) formally defined on the projection function, and requiring a given role to appear in only one side of a parallel composition. The remaining local type syntax is similar to the one of global types.

Example 2.2 (On causalities in global type). Consider the following global type:

$$
\begin{array}{rcll}
G_{seq} & = & \mathtt{r}_1 \to \mathtt{r}_2 & : \ (x : \mathsf{int})\{\mathsf{tt}\}. \\
& & \mathtt{r}_3 \to \mathtt{r}_4 & : \ (y : \mathsf{int})\{\mathsf{tt}\}.\mathsf{end}
\end{array}
$$

The interaction from $\mathtt{r}_1$ to $\mathtt{r}_2$ and the interaction from $\mathtt{r}_3$ to $\mathtt{r}_4$ are causally unrelated. In fact, due to *distribution*, one cannot enforce $\mathtt{r}_3$ to send y after $\mathtt{r}_2$ has received x (unless additional interactions are introduced, for example between $\mathtt{r}_2$

and $\mathtt{r}_3$). In fact (as in [33, 6, 9, 23, 24, 16]) the global type above specifies the same behaviour as

$$G_{par} \;=\; \mathtt{r}_1 \to \mathtt{r}_2 : (x : \mathsf{int})\{\mathsf{tt}\}.\mathsf{end} \mid \mathtt{r}_3 \to \mathtt{r}_4 : (y : \mathsf{int})\{\mathsf{tt}\}.\mathsf{end}$$

Where interactions are causally unrelated, we will use the parallel global types G_{par} rather than the sequential one G_{seq}.

The global type below, instead, requires the completion of the first interaction before $\mathtt{r}_2$ can send the next message:

$$\begin{array}{lcl} \mathtt{r}_1 \to \mathtt{r}_2 & : & (x : \mathsf{int})\{\mathsf{tt}\}. \\ \mathtt{r}_2 \to \mathtt{r}_4 & : & (y : \mathsf{int})\{\mathsf{tt}\}.\mathsf{end} \end{array}$$

In addition, due to *asynchrony*, causality may affect the send and receive actions of an interaction in different ways, as shown by the global type below.

$$\begin{array}{lcl} \mathtt{r}_1 \to \mathtt{r}_2 & : & (x : \mathsf{int})\{\mathsf{tt}\}. \\ \mathtt{r}_1 \to \mathtt{r}_4 & : & (y : \mathsf{int})\{\mathsf{tt}\}.\mathsf{end} \end{array}$$

Variable y must be sent by $\mathtt{r}_1$ only after variable x is *sent*, but possibly before x is *received* by $\mathtt{r}_2$.

One can derive a set of local types T_i from a global type G by *endpoint projection*. As in [23, 24], our definition of endpoint projection relies on a merge operator on local types which is useful to coherently assemble the behaviour that a role has in different branches of a global type, as illustrated in Example 2.3.

Example 2.3 (Merging local behaviours). Consider the following global type:

$$\begin{array}{lcll} \mathtt{r}_1 \to \mathtt{r}_2 & : & \{ & l_1(x : \mathsf{int})\{\mathsf{tt}\}.\mathtt{r}_2 \to \mathtt{r}_3 : l_3(x' : \mathsf{int})\{\mathsf{tt}\}.\mathsf{end}, \\ & & & l_2(y : \mathsf{string})\{\mathsf{tt}\}.\mathtt{r}_2 \to \mathtt{r}_3 : l_4(y' : \mathsf{string})\{\mathsf{tt}\}.\mathsf{end}\} \end{array}$$

When defining the local behaviour of $\mathtt{r}_3$ one must take into account that the first communication between $\mathtt{r}_1$ and $\mathtt{r}_2$ is not visible to $\mathtt{r}_3$. From the perspective of $\mathtt{r}_3$ the session will either be described as *either* $\mathtt{r}_2?l_3(x' : \mathsf{int})\{\mathsf{tt}\}.\mathsf{end}$ or $\mathtt{r}_2?l_4(y' : \mathsf{string})\{\mathsf{tt}\}.\mathsf{end}$. The overall behaviour of $\mathtt{r}_3$ is obtained by *merging* the two local types above into one branching as follows:

$$\mathtt{r}_2?\{l_3(x' : \mathsf{int})\{\mathsf{tt}\}.\mathsf{end},\; l_4(y' : \mathsf{string})\{\mathsf{tt}\}.\mathsf{end}\}$$

Definition 2.1. We define the union of two local types as the following partial operator:

1. $T \cup T = T$

2. $\mathtt{r}?l_k\{(x_k:S_k)\{A_k\}.T_k\}_{k\in I} \cup \mathtt{r}?l_k\{(x_k:S_k)\{A_k\}.T_k\}_{k\in J} =$
$\mathtt{r}?l_k\{(x_k:S_k)\{A_k\}.T_k\}_{k\in I\cup J}$ with $I \cap J = \emptyset$

Local types are idempotent w.r.t. $\cup$ and is otherwise undefined when types are not both input types from the same sender. In this case, all possible labels, together with their associated payload, types, assertions and continuations are collected into a single type.

Definition 2.2. Assume all labels are indexed and $l_i = l_j$ if and only if $i = j$. The merge operator $\sqcup$ is a partial operator on local types, and is defined by the following axioms (closed by standard typed contexts):

1. $T \sqcup T = T$
2. $\mathtt{r}?\{l_i(x_i:S_i)\{A_i\}.T_i\}_{i\in I} \sqcup \mathtt{r}?l_j\{(x'_j:S'_j)\{A'_j\}.T'_j\}_{j\in J} =$
$\mathtt{r}?l_k\{(x_k:S_k)\{A_k\}.T_k\}_{k\in I\setminus J} \cup \mathtt{r}?l_k\{(x'_k:S'_k)\{A'_k\}.T'_k\}_{k\in J\setminus I} \cup$
$\mathtt{r}?l_k\{(x_k:S_k)\{A_k \vee A'_k\}.T_k \sqcup T'_k\}_{k\in I\cap J}$
when $\forall k \in I \cap J,\ x_k = x'_k$, and $S_k = S'_k$.

By (1) each local type is idempotent w.r.t. $\sqcup$. Axiom (2) merges two local types receiving messages from a common role, say $\mathtt{r}$. The resulting local type includes the union of the branches having distinguished labels (i.e. in $I \setminus J$ and $J \setminus I$), and integrates the common labels (i.e., in $I \cap J$). When integrating the common labels, axiom (2) makes sure that they have the same sorts (i.e., $S_k = S'_k$). The merge operator in [23, 24] is defined on local types without assertions. We define the predicate of the resulting local type to be the disjunction of the predicates of the local types being merged (i.e., $A_k \vee A'_k$). We motivate this choice via Examples 2.4 and 2.5. The intuition is that, when allowing merging for receiving actions only, the message, from the point of view of the local participant, satisfies a predicate for one of the branches.

Example 2.4 (Merging assertions). Consider the following global type:

$$\mathtt{r}_1 \to \mathtt{r}_2 \ : \ \{ \ l_1(x : \mathsf{int})\{\mathsf{tt}\}.\mathtt{r}_2 \to \mathtt{r}_3 : l_3(x' : \mathsf{int})\{x' > 0\}.\mathsf{end},\ l_2(y : \mathsf{string})\{\mathsf{tt}\}.\mathtt{r}_2 \to \mathtt{r}_3 : l_3(x' : \mathsf{int})\{x' > 10\}.\mathsf{end}\}$$

This scenario differs from the one in Example 2.3 from the fact that $\mathtt{r}_3$ is expecting label l_3 in both branches (hence the behaviours of the common label l_3 need to be integrated). Role $\mathtt{r}_3$ must be able to accept a value for x' that satisfies $x' > 0$ or $x' > 10$ (without knowing which branch between l_1 or l_2 was selected, hence to which assertion $\mathtt{r}_2$ must obey). Therefore we relax the expectation of $\mathtt{r}_3$ to expect either case (i.e., a value satisfying the disjunction of the predicates):

$$\mathtt{r}_2?\{l_3(x' : \mathsf{int})\{x' > 0 \vee x' > 10\}.\mathsf{end}\}$$

When merging two local types, say T_1 and T_2, if none of the axioms in Definition 2.2 applies then we say that T_1 and T_2 are non mergeable. As in [23, 24] we let the local types for sending interactions to be non mergeable (i.e., axiom 2 can only be applied to receive interactions) unless the send interactions to be merged are identical (in which case one can merge by axiom 1). Example 2.5 illustrates the motivation of this choice.

Example 2.5 (Non mergeability of send interactions). In the global type below

$$\begin{array}{rcl} \mathtt{r}_1 \to \mathtt{r}_2 & : & \{ \quad l_1(x : \mathsf{int})\{\mathsf{tt}\}.\mathtt{r}_3 \to \mathtt{r}_2 : l_3(x' : \mathsf{int})\{\mathsf{tt}\}.\mathsf{end}, \\ & & \quad\; l_2(y : \mathsf{string})\{\mathsf{tt}\}.\mathtt{r}_3 \to \mathtt{r}_2 : l_4(y' : \mathsf{string})\{\mathsf{tt}\}.\mathsf{end}\} \end{array}$$

Considering local behaviour of role $\mathtt{r}_3$ that role $\mathtt{r}_3$ must choose between l_3 and l_4 without knowing which branch was chosen by $\mathtt{r}_1$ in the first interaction. If we allowed to merge the two behaviours of $\mathtt{r}_3$ we would also allow, for instance, $\mathtt{r}_3$ to select l_3 after $\mathtt{r}_1$ had selected l_2. In this scenario the behaviour $\mathtt{r}_3$ would not conform to the expectations of $\mathtt{r}_2$ with respect to the global type. In Definition 2.2 we require, instead, that when a sender $\mathtt{r}$ does not know which branch was chosen by other roles in a previous interaction, then $\mathtt{r}$ must act in the same way in all branches. The following global type is, for instance, mergeable by axiom (1) in Definition 2.2:

$$\begin{array}{rcl} \mathtt{r}_1 \to \mathtt{r}_2 & : & \{ \quad l_1(x : \mathsf{int})\{\mathsf{tt}\}.\mathtt{r}_3 \to \mathtt{r}_2 : l_3(x' : \mathsf{int})\{\mathsf{tt}\}.\mathsf{end}, \\ & & \quad\; l_2(y : \mathsf{string})\{\mathsf{tt}\}.\mathtt{r}_3 \to \mathtt{r}_2 : l_3(x' : \mathsf{int})\{\mathsf{tt}\}.\mathsf{end}\} \end{array}$$

Let $\mathsf{roles}(G)$ be the set of roles in G. Formally,

$$\begin{array}{rcl} \mathsf{roles}(\mathtt{r}_1 \to \mathtt{r}_2 : \{l_i(x_i : S_i)\{A_i\}.G_i\}_{i\in I}) & = & \{\mathtt{r}_1, \mathtt{r}_2\} \bigcup_{i\in I} \mathsf{roles}(G_i) \\ \mathsf{roles}(G_1 \mid G_2) & = & \mathsf{roles}(G_1) \cup \mathsf{roles}(G_2) \\ \mathsf{roles}(\mu \mathtt{t}.G) & = & \mathsf{roles}(G) \\ \mathsf{roles}(\mathtt{t}) = \mathsf{roles}(\mathsf{end}) & = & \emptyset \end{array}$$

We next define $ftv(G)$, the set of *free type variables* in G as:

$$ftv(\mathtt{r}_1 \to \mathtt{r}_2 : \{l_i(x_i : S_i)\{A_i\}.G_i\}_{i\in I}) = \cup_{i\in I} ftv(G_i) \qquad ftv(\mathsf{end}) = \emptyset$$
$$ftv(G_1 \mid G_2) = ftv(G_1) \cup ftv(G_2) \qquad ftv(\mu \mathsf{t}.G) = ftv(G) \setminus \{\mathsf{t}\} \qquad ftv(\mathsf{t}) = \mathsf{t}$$

The set $fv(A)$ of free variables occurring in A is defined as follows:

$$fv(\mathsf{tt}) = fv(\mathsf{ff}) = \emptyset \qquad fv(e_1 = e_2) = fv(e_1 < e_2) = fv(e_1) \cup fv(e_2)$$
$$fv(\neg A) = fv(A) \qquad fv(A_1 \wedge A_2) = fv(A_1 \vee A_2) = fv(A_1) \cup fv(A_2)$$
$$fv(x) = \{x\}$$
$$fv(e_1 \ \mathtt{op} \ e_2) = fv(e_1) \cup fv(e_2) \qquad \mathtt{op} \in \{+, -, *, \mathtt{mod}\}$$

Definition 2.3 (Projection). Assume $\mathtt{r}_1, \mathtt{r}_2, \mathtt{r} \in G$ and $\mathtt{r}_1 \neq \mathtt{r}_2$. The projection of G on $\mathtt{r}$, written $G \upharpoonright \mathtt{r}$, is defined as follows:

$$(\mathtt{r}_1 \to \mathtt{r}_2 : \{l_i(x_i{:}S_i)\{A_i\}.G_i\}_{i\in I}) \upharpoonright \mathtt{r} = \begin{cases} \mathtt{r}_2!\{l_i(x_i : S_i)\{A_i\}.(G_i \upharpoonright \mathtt{r})\}_{i\in I} & \text{if } \mathtt{r} = \mathtt{r}_1 \\ \mathtt{r}_1?\{l_i(x_i : S_i)\{A_i\}.(G_i \upharpoonright \mathtt{r})\}_{i\in I} & \text{if } \mathtt{r} = \mathtt{r}_2 \\ \sqcup_{i\in I} G_i \upharpoonright \mathtt{r} & \{\mathtt{r}_1, \mathtt{r}_2\} \cap \mathsf{roles}(G_i) \neq \emptyset \\ G_i & G_i = \mathsf{t} \text{ or } G_i = \mathsf{end} \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$(G_1 \mid G_2) \upharpoonright \mathtt{r} = \begin{cases} G_i \upharpoonright \mathtt{r} & \text{if } \mathsf{roles}(G_1) \cap \mathsf{roles}(G_2) = \emptyset \\ & \text{and } \mathit{ftv}(G_1) \cap \mathit{ftv}(G_2) = \emptyset \\ & i \in \{1,2\} \text{ and } \mathtt{r} \notin \mathsf{roles}(G_{3-i}) \\ \text{undefined} & \text{otherwise.} \end{cases}$$

$$\mu\mathsf{t}.G \upharpoonright \mathtt{r} = \begin{cases} \mu\mathsf{t}.(G \upharpoonright \mathtt{r}) & \text{if } \mathtt{r} \in G \\ \mathsf{end} & \text{otherwise} \end{cases}$$

$$\mathsf{t} \upharpoonright \mathtt{r} = \mathsf{t}$$

$$\mathsf{end} \upharpoonright \mathtt{r} = \mathsf{end}$$

The first rule projects an interaction onto sender or receiver role. Note that, if the role is not involved in the interaction ($\mathtt{r} \neq \mathtt{r}_2 \neq \mathtt{r}_1$) then the projection is the local type resulting by merging (Definition 2.2) the projections $G_i \upharpoonright \mathtt{r}$ for all $i \in I$. The side condition ensures that we write a parallel composition if the roles and type variables are disjoint (cf. Example 2.2). If some of the $G_i \upharpoonright \mathtt{r}$ are non mergeable then the projection rule cannot be applied. $(G_1 \mid G_2) \upharpoonright \mathtt{r}$ is defined only when the sets of roles of G_1 and G_2 are disjoint; in this case, the result of the projection is the projection of the side of the parallel composition in which $\mathtt{r}$ appears (if $\mathtt{r}$ does not appear further, then the projection of any side can only yield end). The other rules are straightforward.

If none of the rules in Definition 2.3 can be applied on a global type G then G is *not projectable*.

Definition 2.4 (Projectability). A global type G is *projectable* if all of its projections to every role $\mathtt{r} \in \mathsf{roles}(G)$ are defined by Definition 2.3.

Hereafter in this article we will consider only projectable global types.

*Example 2.6 (*ATM*: the local type of* $\mathtt{C}$*).* We present the *local type* $T_{\mathtt{C}}$ obtained by

projecting $G_{\texttt{ATM}}$ on role C.

$$T_{\texttt{C}} = \texttt{A}!\{\texttt{Login}(x_i : \mathsf{string})\{\mathsf{tt}\}.\ \texttt{A}?\{\texttt{LoginOK}()\{\mathsf{tt}\}.\ T_{\texttt{Loop}}\ \texttt{LoginFail}()\{\mathsf{tt}\}.\ \mathsf{end}\}\}$$

$$T_{\texttt{Loop}} = \mu\,\texttt{LOOP}.\ \texttt{S}?\{\texttt{Account}(x_b : \mathsf{int})\{x_b \geq 0\}.\ \texttt{S}!\{\texttt{Withdraw}(x_p : \mathsf{int})\{x_p > 0 \wedge x_b - x_p \geq 0\}.\ \texttt{LOOP},\ \texttt{Deposit}(x_d : \mathsf{int})\{x_d > 0\}.\texttt{LOOP},\ \texttt{Quit}()\{\mathsf{tt}\}.\mathsf{end}\}\}$$

$T_{\texttt{C}}$ specifies the behaviour that C should follow to meet the contract of global type $G_{\texttt{ATM}}$. $T_{\texttt{C}}$ states that C should first authenticate with A, then receive the message Account from S, and then has the choice of sending Withdraw, or Deposit or Quit. If label Withdraw or Deposit are chased another iteration is executed, otherwise the session terminates.

2.2. Monitorability of Global Types

When designing a global type to be used in a monitoring framework, one must ensure that the monitor associated to each role is always able to determine if an incoming or outgoing message conforms to the contract or not. Example 2.7 shows that this is not the case for some global types.

Example 2.7 (Non monitorable global type). In the global type below $\texttt{r}_3$ does not know which value has been given to x in the first interaction between $\texttt{r}_1$ and $\texttt{r}_2$.

$$\begin{array}{rcl} G_s & = & \texttt{r}_1 \to \texttt{r}_2 \ : \ (x : \mathsf{int})\{x > 5\}. \\ & & \texttt{r}_2 \to \texttt{r}_3 \ : \ (y : \mathsf{int})\{\mathsf{tt}\}. \\ & & \texttt{r}_3 \to \texttt{r}_1 \ : \ (z : \mathsf{int})\{x > z\}.\mathsf{end} \end{array}$$

For any value sent by $\texttt{r}_3$, the monitor of $\texttt{r}_3$ cannot determine whether the value sent for z by $\texttt{r}_3$ is violating or not. Similarly (but for receive interactions) in the global type below

$$\begin{array}{rcl} G_r & = & \texttt{r}_1 \to \texttt{r}_2 \ : \ (x : \mathsf{int})\{x > 5\}. \\ & & \texttt{r}_2 \to \texttt{r}_4 \ : \ (y : \mathsf{int})\{y > x\}.\mathsf{end} \end{array}$$

the monitor of $\texttt{r}_4$ will not have, at run-time, information on the value of variable x, hence will not be able to determine if the value sent by $\texttt{r}_2$ for y conforms to the assertion $y > x$.

We call global types as the ones illustrated in Example 2.7 *non monitorable*. In the rest of this section we will give a formal definition of monitorability.

Definition 2.5 (Known variables). Let G'' be a subterm of G. We say that p *knows* x in G'' if:

- there exists G' subterm of G s.t. $G' = \texttt{r}_1 \to \texttt{r}_2 : \{l_i(x_i : S_i)\{A_i\}.G_i\}_{i \in I}$,

- $\mathtt{p} \in \{\mathtt{r}_1, \mathtt{r}_2\}$,
- for some $j \in I$, G'' is a subterm of G_j and $x = x_j$.

Namely, p knows a variable x in a subterm G'' of G if x is introduced in an interaction of G that occurs before G'' and that involves p.

Definition 2.6 (Monitorability). Let G be a subterm of G_0. G is monitorable w.r.t. G_0 if one of the following conditions holds:

1. $G = \mathtt{r}_1 \to \mathtt{r}_2 : \{l_i(x_i : \mathsf{int}_i)\{A_i\}.G_i\}_{i \in I}$, and for all $i \in I$, $y \in fv(A_i)$, $j \in \{1, 2\}$, $\mathtt{r}_j$ knows y in G and G_i is monitorable w.r.t. G_0;
2. $G = G_1 | G_2$, and for all $j \in \{1, 2\}$, G_j is monitorable w.r.t. G_0;
3. $G = \mu \mathtt{t}.G'$ and G' is monitorable w.r.t. G_0;
4. $G = \mathtt{t}$ or $G = \mathsf{end}$.

We say that G is monitorable if it is monitorable w.r.t. G.

Proposition 2.7 (Decidability). Let G and G_0 be global types with G subterm of G_0 and $\mathtt{p} \in \mathsf{roles}(G_0)$. It is decidable if:

1. p knows x in G,
2. G is monitorable w.r.t. G_0.

Proof. (1) follows directly from: (i) the finiteness of the number of subterms of G_0 (and G), (ii) the finiteness of the number of labels in a branching (i.e., the cardinality of the set I of indices), and (iii) the decidability of inclusion in finite sets (e.g., $\mathtt{p} \in \{\mathtt{r}_1, \mathtt{r}_2\}$ and $x \in \{x_i \mid i \in I\}$ in Definition 2.5). Proposition 2.7(2) follows from: (i) the finiteness of the number of subterms of G, (ii) the finiteness of the number of variables in assertions, and (iii) the decidability of p knows x in G by (1).

Knowledge of a name requires a linear search in the prefix. Monitorability requires a quadratic exploration.

In the following sections we will show that, under the assumption that all underlying global types are *projectable and monitorable*, the runtime monitoring discipline we propose ensures that the interactions in a session are safe (e.g., a principal implementing a role never receives messages of unexpected type), and predictable (i.e., faithful to the global interaction pattern specified by the protocol).

Monitorability strengthens a similar property called history sensitivity in [9]. By history sensitivity, only the sender of an interaction must know the free variables of a predicate annotating that interaction. For instance, referring to Example 2.7, G_s is not history sensitive whereas G_r is. In [9] where only static

$$
\begin{array}{rcl}
P & ::= & \overline{a}\langle s[\mathtt{r}] : T\rangle \mid a(y[\mathtt{r}]{:}T).P \mid k[\mathtt{r}_1, \mathtt{r}_2]!l\langle e\rangle \mid k[\mathtt{r}_1, \mathtt{r}_2]?\{l_i(x_i).P_i\}_{i\in I} \mid \\
 & & \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \mid P \mid Q \mid \mathbf{0} \mid \mu X.P \mid X \mid P; Q \mid (\nu a)\ P \mid (\nu s)P \\
N & ::= & [P]_\alpha \mid N_1 \mid N_2 \mid \mathbf{0} \mid (\nu a)N \mid (\nu s)N \mid \langle r\ ;\ h\rangle \\
r & ::= & \emptyset \mid r, s[\mathtt{r}] \mapsto \alpha \qquad h ::= \emptyset \mid h \cdot m \qquad m ::= \overline{a}\langle s[\mathtt{r}] : T\rangle \mid s\langle \mathtt{r}_1, \mathtt{r}_2, l\langle v\rangle\rangle
\end{array}
$$

$\mathtt{r}, \mathtt{r}_1, ...$	roles	$s, s', ...$	session names	$X, Y, ...$	process variables
$a, b, ...$	shared names	$x, y, ...$	variables	$P, Q, ...$	processes
$\alpha, \beta, ...$	principal names	$N, N', ...$	networks		

Figure 4: Processes and the network: syntax

verification is used, it is sufficient to check that all roles *send* values satisfying the assertions. Receivers can rely on this fact thanks to the assumption that the processes implementing the other roles are well-typed. In the run-time verification scenario we cannot assume that the rest of the network behaves safely, hence both sent and received values must be checked. The requirement posed by monitorability, on the other hand, allows our theory to work using a logic without existential quantifiers. On the contrary, in [9] quantifiers were needed, during endpoint projection, to close the assertions w.r.t. those variables that were unknown to the receivers.

3. Formal Framework of Processes and Networks

In this section we introduce a novel *monitored session calculus* as a variant of the π-calculus, which we use to model global networks. Global networks consists of monitors and distributed programs, run by principals and implementing some protocols.

In our formal framework, each distributed application consists of one or more sessions among *principals*. A principal with behaviour P and name α is represented as $[P]_\alpha$. A *network* is a set of principals together with a (unique) *global transport*, which abstractly represents the communication functionality of a distributed system. The syntax of processes, principals, and networks is given in Fig. 4, building on the multiparty session π-calculus from [6].

Processes. Processes, defined in Fig. 4, are ranged over by $P, P', \ldots$ and communicate using two types of channel: *shared channels* (or shared names) used by processes for sending and receiving invitations to participate in sessions, and *session channels* (or session names) used for communication *within* established sessions. Each shared name, say a, is associated to one principal, say α, in the sense that α can read from a; a is shared in the sense that many other principals

can send messages to α through a. One may consider shared names as e.g., URLs or service names. The *session invitation* $\overline{a}\langle s[\mathtt{r}] : T\rangle$ invites, through a shared name a, another process to play $\mathtt{r}$ in a session s. The *session accept* $a(y[\mathtt{r}]:T).P$ receives a session invitation and, after instantiating y with the received session name, behaves in its continuation P as specified by local type T for role $\mathtt{r}$. The *selection* $k[\mathtt{r}_1, \mathtt{r}_2]!l\langle e\rangle$ sends, through session channel k (of an established session), and as a sender $\mathtt{r}_1$ and to a receiver $\mathtt{r}_2$, an expression e with label l. The *branching* $k[\mathtt{r}_1, \mathtt{r}_2]?\{l_i(x_i).P_i\}_{i\in I}$ is ready to receive one of the labels and a value, then behaves as P_i after instantiating x_i with the received value. We omit labels when I is a singleton. The *conditional*, *parallel* and *inaction* are standard. The *recursion* $\mu X.P$ defines X as P. Processes $(\nu a)P$ and $(\nu s)P$ hide shared names and session names, respectively.

Principals and Network. Principals and networks are also formally defined in Fig. 4. A *principal* $[P]_\alpha$, with process P and name α, represents a unit of behaviour (hence verification) in a distributed system. A *network* N is a collection of principals with a unique global transport. The behaviour of a principal, described in its process, includes communication over shared channels to create or join new sessions, the communication over session channels, and control structures such as conditional branching and recursion.

A *global transport* is a pair $\langle r \ ; \ h\rangle$ of a routing table r that associates roles to principals, and a global queue h. The *routing table* r is a finite map from session-roles and shared names to principals. If, for instance, $r(a) = \alpha$ then a session invitation message through a will be delivered to principal α. Similarly, if $r(s[\mathtt{r}]) = \alpha$ then a message for $\mathtt{r}$ in session s will be delivered to principal α. The *global queue* h is a sequence of *messages* $\overline{a}\langle s[\mathtt{r}] : T\rangle$ or $s\langle \mathtt{r}_1, \mathtt{r}_2, l\langle v\rangle\rangle$, ranged over by m. These m represent messages-in-transit, i.e. those messages which have been sent by some principal but have not yet been delivered. Possible shuffles changing the ordering of in-transit messages is discussed below. Networks are composed of principals and global transport.

Let $n, n', \ldots$ range over shared and session channels. A network N that satisfies the following conditions is *well-formed*: (1) N contains at most one global transport; (2) two principals in N never have the same principal name; and (3) if $N \equiv (\nu\tilde{n})(\prod_i [P_i]_{\alpha_i} | \langle r \ ; \ h\rangle)$ then each free shared or session name in P_i and h occurs in $\tilde{n}$ (we use $\prod_i P_i$ to denote $P_1 \mid P_2 \cdots \mid P_n$).

Semantics. The reduction relation for networks is generated from the rules defined in Fig. 5, which model the interactions of principals with the global queue. Rule $\lfloor$REQ$\rfloor$ places an invitation to participate as role $\mathtt{r}$ in session s into the global queue. Dually, in $\lfloor$ACC$\rfloor$, a process receives an invitation on a shared name from the global queue, assuming a message on a is to be routed to α. As a result, the

$$[\overline{a}\langle s[\mathtt{r}] : T\rangle]_\alpha \mid \langle r \,;\, h\rangle \longrightarrow [\mathbf{0}]_\alpha \mid \langle r \,;\, h \cdot \overline{a}\langle s[\mathtt{r}] : T\rangle\rangle \qquad \lfloor \text{REQ} \rfloor$$

$$[a(y[\mathtt{r}] : T).P]_\alpha \mid \langle r \,;\, \overline{a}\langle s[\mathtt{r}] : T\rangle \cdot h\rangle \longrightarrow [P[s/y]]_\alpha \mid \langle r, s[\mathtt{r}] \mapsto \alpha \,;\, h\rangle^\dagger \qquad \lfloor \text{ACC} \rfloor$$

$$[s[\mathtt{r}_1, \mathtt{r}_2]!l_j\langle v\rangle]_\alpha \mid \langle r \,;\, h\rangle \longrightarrow [\mathbf{0}]_\alpha \mid \langle r \,;\, h \cdot s\langle \mathtt{r}_1, \mathtt{r}_2, l_j\langle v\rangle\rangle\rangle \qquad \lfloor \text{SEL} \rfloor$$

$$[s[\mathtt{r}_1, \mathtt{r}_2]?\{l_i(x_i).P_i\}_i]_\alpha \mid \langle r \,;\, s\langle \mathtt{r}_1, \mathtt{r}_2, l_j\langle v\rangle\rangle \cdot h\rangle \longrightarrow [P_j[v/x_j]]_\alpha \mid \langle r \,;\, h\rangle^{\dagger\dagger} \qquad \lfloor \text{BRA} \rfloor$$

$$[\mathsf{if}\ \mathsf{tt}\ \mathsf{then}\ P\ \mathsf{else}\ Q]_\alpha \longrightarrow [P]_\alpha \qquad [\mathsf{if}\ \mathsf{ff}\ \mathsf{then}\ P\ \mathsf{else}\ Q]_\alpha \longrightarrow [Q]_\alpha \qquad \lfloor \text{CND} \rfloor$$

$$\frac{[P]_\alpha \mid N \longrightarrow [P']_\alpha \mid N'}{[\mathcal{E}(P)]_\alpha \mid N \longrightarrow [\mathcal{E}(P')]_\alpha \mid N'} \quad \frac{e \longrightarrow e'}{[\mathcal{E}(e)]_\alpha \longrightarrow [\mathcal{E}(e')]_\alpha} \quad \frac{N \longrightarrow N'}{\mathcal{E}(N) \longrightarrow \mathcal{E}(N')} \qquad \lfloor \text{CTX} \rfloor$$

$$\dagger : r(a) = \alpha \qquad \dagger\dagger : r(s[\mathtt{r}_2]) = \alpha$$

$$\mathcal{E} ::= \quad (\,) \mid \mathcal{E} \mid P \mid (\nu s)\mathcal{E} \mid (\nu a)\mathcal{E} \mid \mathcal{E}; P \mid \mathcal{E} \mid N \mid \text{if } \mathcal{E} \text{ then } P \text{ else } Q \mid s[\mathtt{r}_1, \mathtt{r}_2]!l\langle \mathcal{E}\rangle$$

Figure 5: Reduction for dynamic networks

routing table adds $s[\mathtt{r}] \mapsto \alpha$ in the entry for s. Rule $\lfloor$SEL$\rfloor$ puts in the queue a message sent from $\mathtt{r}_1$ to $\mathtt{r}_2$, which selects label l_j and carries v, if it is not going to be routed to α (i.e. sent to self). Dually, $\lfloor$BRA$\rfloor$ gets a message with label l_j from the global queue, so that the j-th process P_j receives value v. Rules $\lfloor$CTX$\rfloor$ are for a closure under the reduction context $\mathcal{E}$. The other rules are standard.

The reduction is also defined modulo the structural congruence $\equiv$ defined by the standard laws over processes/networks, the unfolding of recursion ($\mu X.P \equiv P[\mu X.P/X]$) and the associativity and commutativity and the rules of message permutation in the queue [33, 23]. The rules are summarised in Fig. 6 where $u \in \{s, a\}$. The rule for message permutation is

$$\frac{m_1 \cdot m_2 \curvearrowright m_2 \cdot m_1}{h \cdot m_1 \cdot m_2 \cdot h' \equiv h \cdot m_2 \cdot m_1 \cdot h'}$$

and uses the notion of message permutation given in Definition 3.1. The rule for message permutation is needed to treat the inherent non-determinism arising in message orders when three or more participants are involved. The other rules are straightforward.

Definition 3.1 (Message permutation). For messages in the global queue, we say messages m_1 and m_2 are *permutable*, denoted by $m_1 \cdot m_2 \curvearrowright m_2 \cdot m_1$, if they satisfy one of the following conditions:

1. $m_1 = s\langle \mathtt{r}_1, \mathtt{r}_2, l\langle v\rangle\rangle$ and $m_2 = s'\langle \mathtt{r}'_1, \mathtt{r}'_2, l'\langle v'\rangle\rangle$, where $s \neq s'$, or ($\mathtt{r}_1 \neq \mathtt{r}'_1 \ \wedge \ \mathtt{r}_2 \neq \mathtt{r}'_2$).
2. $m_1 = \overline{a}\langle s[\mathtt{r}] : T\rangle$ and $m_2 = \overline{a'}\langle s'[\mathtt{r}'] : T'\rangle$, where $a \neq a'$.
3. $m_i = \overline{a}\langle s[\mathtt{r}] : T\rangle$ and $m_j = s'\langle \mathtt{r}', \mathtt{r}'', l\langle v\rangle\rangle$ and $i, j \in \{1, 2\}, i \neq j$, where $s \neq s'$, or ($\mathtt{r} \neq \mathtt{r}' \ \wedge \ \mathtt{r} \neq \mathtt{r}''$).

$$P \mid \mathbf{0} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R)$$

$$(\nu\, u)P \mid Q \equiv (\nu\, u)(P \mid Q) \text{ if } u \in \{s, a\}, u \notin \mathsf{fn}(Q) \qquad (\nu\, uu')P \equiv (\nu\, u'u)P$$

$$(\nu\, u)\mathbf{0} \equiv \mathbf{0} \qquad \frac{P \equiv Q}{[P]_\alpha \equiv [Q]_\alpha} \qquad (\nu\, u)[\mathbf{0}]_\alpha \equiv [\mathbf{0}]_\alpha$$

$$N \mid [\mathbf{0}]_\alpha \equiv N \qquad N_1 \mid N_2 \equiv N_2 \mid N_1 \qquad (N_1 \mid N_2) \mid N_3 \equiv N_1 \mid (N_2 \mid N_3)$$

$$(\nu\, u)N_1 \mid N_2 \equiv (\nu\, u)(N_1 \mid N_2) \text{ if } u \notin \mathsf{fn}(N_2) \qquad (\nu\, uu')N \equiv (\nu\, u'u)N$$

$$\emptyset \cdot h \equiv h \qquad \frac{m_1 \cdot m_2 \curvearrowright m_2 \cdot m_1}{h \cdot m_1 \cdot m_2 \cdot h' \equiv h \cdot m_2 \cdot m_1 \cdot h'} \qquad \frac{r = r \quad h \equiv h'}{\langle r \,;\, h \rangle \equiv \langle r' \,;\, h' \rangle}$$

Figure 6: Structural congruence for networks

By (1) two interaction messages (i.e., in ongoing sessions) are permutable if they are not related to the same session, or they are related to the same session but their roles are different. By (2) two invitation messages are permutable if they are for different principals. By (3) an invitation message and an interaction message are permutable if they are for different sessions or the invited role $\mathtt{r}$ in the invitation message is different from both the sender and receiver of the interaction message.

*Example 3.1 (*ATM*: an implementation).* We now illustrate the processes implementing the client role of the *ATM* protocol. We let $P_{\mathtt{C}}$ be the process implementing $T_{\mathtt{C}}$ (from Example 2.6) and communicating on session channel s.

$$\begin{array}{l|l} P_{\mathtt{C}} = s[\mathtt{C},\mathtt{A}]!\mathtt{Login}(\text{“}alice\_pwd123\text{”}); & P''_{\mathtt{C}} = \mathsf{if}\ getmore() \wedge (x_b \geq 10) \\ \qquad s[\mathtt{A},\mathtt{C}]?\{\mathtt{LoginOK}(); \mu X.P'_{\mathtt{C}}, \mathtt{LoginFail}().\mathbf{0}\} & \qquad \mathsf{then}\ s[\mathtt{C},\mathtt{S}]!\mathtt{Withdraw}(10); X \\ P'_{\mathtt{C}} = s[\mathtt{S},\mathtt{C}]?\mathtt{Account}(x_b); P''_{\mathtt{C}} & \qquad \mathsf{else}\ s[\mathtt{C},\mathtt{S}]!\mathtt{Quit}(); \mathbf{0} \end{array}$$

Note that $P_{\mathtt{C}}$ selects only two of the possible branches (i.e., `Withdraw` and `Quit`) and `Deposit` is never selected. One can think of $P_{\mathtt{C}}$ as an ATM machine that only allows to withdraw a number of £10 banknotes, until the amount exceeds the current balance. This ATM machine does not allow deposits. We assume $getmore()$ to be a local function to the principal running $P_{\mathtt{C}}$ that returns tt if more notes are required, and ff otherwise. $P_{\mathtt{S}}$ below implements the server role:

$$\begin{array}{l|l} P_{\mathtt{S}} = s[\mathtt{A},\mathtt{S}]?\{\mathtt{LoginOK}(); \mu X.P'_{\mathtt{S}}, \mathtt{LoginFail}().\mathbf{0}\} & P''_{\mathtt{S}} = s[\mathtt{C},\mathtt{S}]?\{\mathtt{Withdraw}(x_p).X, \\ P'_{\mathtt{S}} = s[\mathtt{S},\mathtt{C}]!\mathtt{Account}(getBalance()); P''_{\mathtt{S}} & \qquad \mathtt{Deposit}(x_d).X, \\ & \qquad \mathtt{Quit}().\mathbf{0}\ \} \end{array}$$

We assume that $getBalance()$ is a function, local to the principal running $P_{\mathtt{S}}$, that synchronously returns the current balance of the client.

4. The Monitored Network: Semantics and Equivalences

In this section we formalise the specifications (based on local types) used to guard the runtime behaviour of the principals in a network. These specifications are the foundation of system monitors, each wrapping a principal to ensure that the ongoing communication conforms to the given specification. Then, we present a behavioural theory for monitored networks and their safety properties.

4.1. Semantics of Global Specifications

The specification of the (correct) behaviour of a principal consists of an *assertion environment* $\langle \Gamma; \Delta \rangle$, where Γ is the *shared environment* describing the behaviour on shared channels, and Δ is the *session environment* representing the behaviour on session channels (i.e., describing the sessions that the principal is currently participating in). The syntax of Γ and Δ is given by:

$$\Gamma \quad ::= \quad \emptyset \mid \Gamma, a : \mathtt{I}(T[\mathtt{r}]) \mid \Gamma, a : \mathtt{O}(T[\mathtt{r}]) \qquad\qquad \Delta \quad ::= \quad \emptyset \mid \Delta, s[\mathtt{r}] : T$$

In Γ, the assignment $a : \mathtt{I}(T[\mathtt{r}])$ (resp. $a : \mathtt{O}(T[\mathtt{r}])$) states that the principal can, through a, receive (resp. send) invitations to play role $\mathtt{r}$ in a session instance specified by T. In Δ, we write $s[\mathtt{r}] : T$ when the principal is playing role $\mathtt{r}$ of session s specified by T. A network is monitored with respect to *collections of specifications* (later, just *specifications*) one for each principal in the network. A specification $\Sigma, \Sigma', \ldots$ is a finite map from principals to assertion environments:

$$\Sigma \quad ::= \quad \emptyset \mid \Sigma, \alpha : \langle \Gamma; \Delta \rangle$$

The semantics of Σ is defined using the following labels:

$$\ell ::= \overline{a}\langle s[\mathtt{r}] : T \rangle \mid a\langle s[\mathtt{r}] : T \rangle \mid s[\mathtt{r}_1, \mathtt{r}_2]!l\langle v \rangle \mid s[\mathtt{r}_1, \mathtt{r}_2]?l\langle v \rangle \mid \tau$$

The first two labels are for *invitation* actions, the first is for requesting and the second is for accepting. Labels with $s[\mathtt{r}_1, \mathtt{r}_2]$ indicate *interaction* actions for sending (!) or receiving (?) messages within sessions. The labelled transition relation for specifications is defined by the rules in Fig. 7 Rule [Req] allows α to send an invitation on a properly typed shared channel a (i.e., given that the shared environment maps a to $T[\mathtt{r}]$). Rule [Acc] allows α to receive an invitation to be role $\mathtt{r}$ in a new session s, on a properly typed shared channel a. Rule [Bra] allows α, participating to session s as $\mathtt{r}_2$, to receive a message with label l_j from $\mathtt{r}_1$, given that A_j is satisfied after replacing x_j with the received value v. After the application of this rule the specification is T_j. Rule [Sel] is the symmetric (output) counterpart of [Bra]. We use $\downarrow$ to denote the evaluation of a logical assertion. [Spl] is the juxtaposition of two session environments. [Tau] says that the specification should be invariant under reduction of principals. [Par] says if Σ_1 and Σ_3 are composable, after Σ_1 becomes as Σ_2, they are still composable.

$$\alpha : \langle \Gamma, a : \mathtt{0}(T[\mathtt{r}]); \Delta \rangle \xrightarrow{\overline{a}\langle s[\mathtt{r}]:T \rangle} \alpha : \langle \Gamma, a : \mathtt{0}(T[\mathtt{r}]); \Delta \rangle \qquad [\textsc{Req}]$$

$$\frac{s \notin \mathsf{dom}(\Delta)}{\alpha : \langle \Gamma, a : \mathtt{I}(T[\mathtt{r}]); \Delta \rangle \xrightarrow{a\langle s[\mathtt{r}]:T \rangle} \alpha : \langle \Gamma, a : \mathtt{I}(T[\mathtt{r}]); \Delta, s[\mathtt{r}] : T \rangle} \qquad [\textsc{Acc}]$$

$$\frac{\Gamma \vdash v : S_j, \quad A_j[v/x_j] \downarrow \mathsf{tt}, \quad j \in I}{\alpha : \langle \Gamma; \Delta, s[\mathtt{r}_2] : \mathtt{r}_1 ? \{ l_i(x_i : S_i)\{A_i\}.T_i \}_{i \in I} \rangle \xrightarrow{s[\mathtt{r}_1, \mathtt{r}_2] ? l_j \langle v \rangle} \alpha : \langle \Gamma; \Delta, s[\mathtt{r}_2] : T_j[v/x_j] \rangle} \qquad [\textsc{Bra}]$$

$$\frac{\Gamma \vdash v : S_j, \quad A_j[v/x_j] \downarrow \mathsf{tt}, \quad j \in I}{\alpha : \langle \Gamma; \Delta, s[\mathtt{r}_1] : \mathtt{r}_2 ! \{ l_i(x_i : S_i)\{A_i\}.T_i \}_{i \in I} \rangle \xrightarrow{s[\mathtt{r}_1, \mathtt{r}_2] ! l_j \langle v \rangle} \alpha : \langle \Gamma; \Delta, s[\mathtt{r}_1] : T_j[v/x_j] \rangle} \qquad [\textsc{Sel}]$$

$$\frac{\alpha : \langle \Gamma_1; \Delta_1 \rangle \xrightarrow{\ell} \alpha : \langle \Gamma_1'; \Delta_1' \rangle}{\alpha : \langle \Gamma_1; \Delta_1, \Delta_2 \rangle \xrightarrow{\ell} \alpha : \langle \Gamma_1'; \Delta_1', \Delta_2 \rangle} \qquad \Sigma \xrightarrow{\tau} \Sigma \qquad \frac{\Sigma_1 \xrightarrow{\ell} \Sigma_2}{\Sigma_1, \Sigma_3 \xrightarrow{\ell} \Sigma_2, \Sigma_3} \qquad [\textsc{Spl},\textsc{Tau},\textsc{Par}]$$

Figure 7: Labelled transition relation for specifications

4.2. Semantics of Dynamic Monitoring

The endpoint monitor $\mathsf{M}, \mathsf{M}', ...$ for principal α is a specification $\alpha : \langle \Gamma; \Delta \rangle$ used to *dynamically* ensure that the messages to and from α are legal with respect to Γ and Δ. A *monitored network* N is a network N with monitors, obtained by extending the syntax of networks as:

$$\mathsf{N} \ ::= \ N \mid \mathsf{M} \quad \mid \quad \mathsf{N} \mid \mathsf{N} \quad \mid \quad (\nu s)\mathsf{N} \quad \mid \quad (\nu a)\mathsf{N}$$

The reduction rules for monitored networks are given in Fig. 8 and use, in the premises, the labelled transitions of monitors. The labelled transitions of a monitor are the labelled transitions of its corresponding specification (given in § 4.1).

The first four rules model reductions that are allowed by the monitor (i.e., in the premise). Rule ⌈Req⌉ inserts an invitation in the global queue. Rule ⌈Acc⌉ is symmetric and updates the router so that all messages for role $\mathtt{r}$ in session s will be routed to α. Similarly, ⌈Bra⌉ (resp. ⌈Sel⌉) extracts (resp. introduces) messages from (resp. in) the global queue. The error cases for ⌈Req⌉ and ⌈Sel⌉, namely ⌈ReqEr⌉ and ⌈SelEr⌉, 'skip' the current action (removing it from the process), and do not modify the queue, the router nor the state of the monitor. The error cases for ⌈Acc⌉ and ⌈Bra⌉, namely ⌈AccEr⌉ and ⌈BraEr⌉, do not affect the process, which remains ready to perform the action, and remove the violating message from the queue.

Example 4.1 (ATM: a monitored network). We illustrate the monitored network for the ATM scenario, where the routing table is defined as

$$r = a \mapsto \alpha, b \mapsto \beta, c \mapsto \gamma, s[\mathtt{S}] \mapsto \alpha, s[\mathtt{C}] \mapsto \beta, s[\mathtt{A}] \mapsto \gamma$$

$$\lceil \textsc{Req} \rceil \frac{\mathsf{M} \xrightarrow{\overline{a}\langle s[\mathtt{r}]:T\rangle} \mathsf{M}'}{[\overline{a}\langle s[\mathtt{r}]:T\rangle]_\alpha \mid \mathsf{M} \mid \langle r\ ;\ h\rangle \longrightarrow [\mathbf{0}]_\alpha \mid \mathsf{M}' \mid \langle r\ ;\ h \cdot \overline{a}\langle s[\mathtt{r}]:T\rangle\rangle}$$

$$\lceil \textsc{Acc} \rceil \frac{\mathsf{M} \xrightarrow{a\langle s[\mathtt{r}]:T\rangle} \mathsf{M}' \qquad r(a) = \alpha}{[a(y[\mathtt{r}]:T).P]_\alpha \mid \mathsf{M} \mid \langle r\ ;\ \overline{a}\langle s[\mathtt{r}]:T\rangle \cdot h\rangle \longrightarrow [P[s/y]]_\alpha \mid \mathsf{M}' \mid \langle r{\cdot}s[\mathtt{r}] \mapsto \alpha\ ;\ h\rangle}$$

$$\lceil \textsc{Bra} \rceil \frac{\mathsf{M} \xrightarrow{s[\mathtt{r}_1,\mathtt{r}_2]?l_j\langle v\rangle} \mathsf{M}' \qquad r(s[\mathtt{r}_2]) = \alpha}{[s[\mathtt{r}_1,\mathtt{r}_2]?\{l_i(x_i).P_i\}_i]_\alpha \mid \mathsf{M} \mid \langle r\ ;\ s\langle \mathtt{r}_1,\mathtt{r}_2,l_j\langle v\rangle\rangle \cdot h\rangle \longrightarrow [P_j[v/x_j]]_\alpha \mid \mathsf{M}' \mid \langle r\ ;\ h\rangle}$$

$$\lceil \textsc{Sel} \rceil \frac{\mathsf{M} \xrightarrow{s[\mathtt{r}_1,\mathtt{r}_2]!l\langle v\rangle} \mathsf{M}' \quad r(s[\mathtt{r}_2]) \neq \alpha}{[s[\mathtt{r}_1,\mathtt{r}_2]!l\langle v\rangle]_\alpha \mid \mathsf{M} \mid \langle r\ ;\ h\rangle \longrightarrow [\mathbf{0}]_\alpha \mid \mathsf{M}' \mid \langle r\ ;\ h \cdot s\langle \mathtt{r}_1,\mathtt{r}_2,l\langle v\rangle\rangle\rangle}$$

$$\lceil \textsc{ReqEr} \rceil \frac{\mathsf{M} \xrightarrow{\overline{a}\langle s[\mathtt{r}]:T\rangle}\!\!\!\!\!\!/\ \ }{[\overline{a}\langle s[\mathtt{r}]:T\rangle]_\alpha \mid \mathsf{M} \mid \langle r\ ;\ h\rangle \longrightarrow [\mathbf{0}]_\alpha \mid \mathsf{M} \mid \langle r\ ;\ h\rangle}$$

$$\lceil \textsc{AccEr} \rceil \frac{\mathsf{M} \xrightarrow{a\langle s[\mathtt{r}]:T\rangle}\!\!\!\!\!\!/\ \ }{[a(y[\mathtt{r}]:T).P]_\alpha \mid \mathsf{M} \mid \langle r\ ;\ \overline{a}\langle s[\mathtt{r}]:T\rangle \cdot h\rangle \longrightarrow [a(y[\mathtt{r}]:T).P]_\alpha \mid \mathsf{M} \mid \langle r\ ;\ h\rangle}$$

$$\lceil \textsc{BraEr} \rceil \frac{\mathsf{M} \overset{\ell}{\not\rightarrow} \quad \ell = s[\mathtt{r}_1,\mathtt{r}_2]?l_j\langle v\rangle}{[s[\mathtt{r}_1,\mathtt{r}_2]?\{l_i(x_i).P_i\}_i]_\alpha \mid \mathsf{M} \mid \langle r\ ;\ s\langle \mathtt{r}_1,\mathtt{r}_2,l_j\langle v\rangle\rangle \cdot h\rangle \longrightarrow [s[\mathtt{r}_1,\mathtt{r}_2]?\{l_i(x_i).P_i\}_i]_\alpha \mid \mathsf{M} \mid \langle r\ ;\ h\rangle}$$

$$\lceil \textsc{SelEr} \rceil \frac{\mathsf{M} \xrightarrow{s[\mathtt{r}_1,\mathtt{r}_2]!l\langle v\rangle}\!\!\!\!\!\!/\ \ }{[s[\mathtt{r}_1,\mathtt{r}_2]!l\langle v\rangle]_\alpha \mid \mathsf{M} \mid \langle r\ ;\ h\rangle \longrightarrow [\mathbf{0}]_\alpha \mid \mathsf{M} \mid \langle r\ ;\ h\rangle}$$

Figure 8: Reduction for monitored networks (assume $\mathsf{M} = \alpha : \langle \Gamma ; \Delta \rangle$).

We consider the fragment of session where the authentication has occurred, the process of $\mathtt{C}$ (resp. $\mathtt{S}$) is $P'_{\mathtt{C}}$ (resp. $P'_{\mathtt{S}}$) from Example 3.1, and the process of $\mathtt{A}$ is $\mathbf{0}$.

$\mathsf{N}_{\mathtt{S}} = [P'_{\mathtt{S}}]_\alpha \mid \mathsf{M}_{\mathtt{S}} = [s[\mathtt{S},\mathtt{C}]!\ \mathtt{Account}\langle 100\rangle; P''_{\mathtt{S}}]_\alpha \mid \mathsf{M}_{\mathtt{S}}$ (assuming $getBalance() = 100$)
$\mathsf{N}_{\mathtt{C}} = [P'_{\mathtt{C}}]_\beta \mid \mathsf{M}_{\mathtt{C}} = [s[\mathtt{S},\mathtt{C}]?\ \mathtt{Account}(x_b).P''_{\mathtt{C}}]_\beta \mid \mathsf{M}_{\mathtt{C}}$
$\mathsf{N}_{\mathtt{A}} = [\mathbf{0}]_\gamma \mid \gamma : \langle c : T_{\mathtt{A}}[\mathtt{A}]\ ;\ s[\mathtt{A}] :\ \mathsf{end}\rangle$

where $\mathsf{M}_{\mathtt{S}} = \alpha : \langle a :\ T_{\mathtt{S}}[\mathtt{S}]\ ;\ s[\mathtt{S}] :\ \mathtt{C}!\ \mathtt{Account}(x_b : \mathsf{int})\{x_b \geq 0\}.T'_{\mathtt{S}}\rangle$ and $\mathsf{M}_{\mathtt{C}}$ is dual.

$$\mathsf{N}_1 = [s[\mathtt{S},\mathtt{C}]!\ \mathtt{Account}\langle 100\rangle; P'_{\mathtt{S}}]_\alpha \mid \mathsf{M}_{\mathtt{S}} \mid [s[\mathtt{S},\mathtt{C}]?\ \mathtt{Account}(x_b).P'_{\mathtt{C}}]_\beta \mid \mathsf{M}_{\mathtt{C}} \mid \mathsf{N}_{\mathtt{A}} \mid \langle r\ ;\ \emptyset\rangle$$
$$\longrightarrow\longrightarrow [P'_{\mathtt{S}}]_\alpha \mid \mathsf{M}'_{\mathtt{S}} \mid [P'_{\mathtt{C}}[100/x_b]]_\beta \mid \mathsf{M}'_{\mathtt{C}} \mid \mathsf{N}_{\mathtt{A}} \mid \langle r\ ;\ \emptyset\rangle$$

where $\mathsf{M}'_{\mathtt{S}} = \alpha : \langle a{:}T_{\mathtt{S}}[\mathtt{S}]\ ;\ s[\mathtt{S}] :\ T'_{\mathtt{S}}\rangle$ and $\mathsf{M}'_{\mathtt{C}} = \beta : \langle b : T_{\mathtt{C}}[\mathtt{C}]\ ;\ s[\mathtt{C}] :\ T'_{\mathtt{C}}\rangle$

Above, $x_b \geq 0$ is satisfied since $x_b = 100$. If the server tried to communicate e.g., value -100 for x_b, then the monitor (by rule $\lceil \textsc{SelEr} \rceil$) would drop the message.

Following Example 4.1, in the example that follows we show the different behaviours of monitored and unmonitored processes.

Example 4.2 (Compare a monitored process to an unmonitored one.). Let

$$\begin{array}{lcl} \ell & = & s[\mathtt{S},\mathtt{C}]!\mathtt{Account}\langle -10\rangle \\ P_1 & = & s[\mathtt{S},\mathtt{C}]!\ \mathtt{Account}\langle -10\rangle; P'_{\mathtt{S}} \\ \mathsf{M}_{\mathtt{S}} & = & \alpha : \langle a :\ T_{\mathtt{S}}[\mathtt{S}]\ ;\ s[\mathtt{S}] :\ \mathtt{C}!\ \mathtt{Account}(x_b : \mathsf{int})\{x_b \geq 0\}.T'_{\mathtt{S}}\rangle \end{array}$$

The unmonitored principal $[P_1]_\alpha$ can make a step ℓ, namely $[P_1]_\alpha \xrightarrow{\ell}$. However, the its monitored counter-part $\mathsf{N}_\mathrm{S} = [P_1]_\alpha \mid \mathsf{M}_\mathrm{S}$ cannot make a step ℓ (that is $\mathsf{N}_\mathrm{S} \not\xrightarrow{\ell}$) since the value -10 does not satisfy the predicate $x_b \geq 0$ attached to the local type of the session monitored by M_S.

Similarly, for type violations, consider:

$$\begin{array}{rcl} \ell & = & s[\mathrm{S},\mathrm{C}]!\mathtt{Account}\langle \text{“}hello\text{”}\rangle \\ P_2 & = & s[\mathrm{S},\mathrm{C}]!\,\mathtt{Account}\langle \text{“}hello\text{”}\rangle; P_\mathrm{S}'' \end{array}$$

then $[P_2]_\alpha \xrightarrow{\ell}$ but $[P_2]_\alpha \mid \mathsf{M}_\mathrm{S} \not\xrightarrow{\ell}$.

4.3. Network Satisfaction and Equivalences

Based on the formal representations of monitored networks, we now introduce the key formal tools for analysing their behaviour. Concretely, we introduce two different equivalences to semantically compare networks: *bisimulation* and *barbed congruence* (the latter relying on the notion of *interface*, also given in this section). The two equivalences allow us to compare networks using different granularities. On the one side, bisimilarity addresses mainly partial networks, and gives an equivalence that distinguishes two networks containing different components. On the other side, (barbed) congruence addresses networks (including global transport) from the point of view of an external observer; thus, two networks built from different components but offering the same service will be equated. We choose to give two equivalences in order to give the theory a way to compare session networks from two different points of views: bisimulation allows designers to equate networks whose structures are similar, whereas barbed congruence allows users to equate networks, seen as black boxes, which provide the same service. Both equivalences are compositional, as proved in Proposition 4.4. Finally, using the definition of congruence, we define *the satisfaction relation* $\models N \triangleright \mathsf{M}$, used in §5 and §7 to prove the properties of our framework.

Bisimulations. We first define semantics for networks of components, or partial networks, on which we define bisimulation: we use $M, M', ...$ for a *partial network*, that is a network without a global transport, hence allowing the global observation of interactions. The labelled transition relation for processes and partial networks M is defined in Fig. 9. In (CTX), $\mathsf{n}(\ell)$ indicates the names occurring in ℓ while $\mathsf{bn}(\mathcal{E})$ indicates binding induced by $\mathcal{E}$. In (RES), $\mathsf{sbj}(\ell)$ denotes the subject of ℓ. In (TAU) the axiom is obtained either from the reduction rules for dynamic networks given in § 3 (only those not involving the global transport), or from the corresponding rules for monitored networks (which have been omitted in § 4.2).

Hereafter we write $\Longrightarrow$ for $\xrightarrow{\tau}{}^*$, $\overset{\ell}{\Longrightarrow}$ for $\Longrightarrow\xrightarrow{\ell}\Longrightarrow$, and $\overset{\hat{\ell}}{\Longrightarrow}$ for $\Longrightarrow$ if $\ell = \tau$ and $\overset{\ell}{\Longrightarrow}$ otherwise.

$$(\text{REQ})\ [\overline{a}\langle s[\mathtt{r}]:T\rangle;P]_\alpha \xrightarrow{\overline{a}\langle s[\mathtt{r}]:T\rangle} [\mathbf{0}]_\alpha \qquad (\text{ACC})\ [a(y[\mathtt{r}]:T).P]_\alpha \xrightarrow{a\langle s[\mathtt{r}]:T\rangle} [P[s/y]]_\alpha$$

$$(\text{BRA})\ [s[\mathtt{r}_1,\mathtt{r}_2]?\{l_i(x_i{:}S_i).P_i\}_i]_\alpha \xrightarrow{s[\mathtt{r}_1,\mathtt{r}_2]?l_j\langle v\rangle} [P_j[v/x_j]]_\alpha$$

$$(\text{SEL})\ [s[\mathtt{r}_1,\mathtt{r}_2]!l_j\langle v\rangle]_\alpha \xrightarrow{s[\mathtt{r}_1,\mathtt{r}_2]!l_j\langle v\rangle} [\mathbf{0}]_\alpha \qquad (\text{CTX})\ \frac{[P]_\alpha \xrightarrow{\ell} [P']_\alpha\ \ \mathsf{n}(\ell)\cap\mathsf{bn}(\mathcal{E})=\emptyset}{[\mathcal{E}(P)]_\alpha \xrightarrow{\ell} [\mathcal{E}(P')]_\alpha}$$

$$(\text{TAU})\ \frac{M \longrightarrow M'}{M \xrightarrow{\tau} M'} \quad (\text{RES})\ \frac{M \xrightarrow{\ell} M'\ a\notin\mathsf{sbj}(\ell)}{(\nu a)M \xrightarrow{\ell\backslash a} (\nu a)M'} \quad (\text{STR})\ \frac{M \equiv M_0 \xrightarrow{\ell} M_0' \equiv M'}{M \xrightarrow{\ell} M'}$$

Figure 9: Labelled transition relation for processes and partial networks

Definition 4.1 (Bisimulation over partial networks). A binary relation $\mathcal{R}$ over partial networks is a *weak bisimulation* when $M_1\mathcal{R}M_2$ implies: whenever $M_1 \xrightarrow{\ell} M_1'$ such that $\mathsf{bn}(\ell)\cap\mathsf{fn}(M_2)=\emptyset$, we have $M_2 \xRightarrow{\hat{\ell}} M_2'$ such that $M_1'\mathcal{R}M_2'$, and the symmetric case. We write $M_1 \approx M_2$ if (M_1, M_2) are in a weak bisimulation.

Interface. As stated above, we build another model where two different implementations of the same service are equated. Bisimilarity is too strong for this aim as shown in further Example 4.3. We therefore introduce a contextual congruence (barbed reduction-closed congruence [32]) $\cong$ for networks. Intuitively, two networks are barbed-congruent when they are indistinguishable for any principal that connects to them. In this case we say that the two (barbed-congruent) networks propose the same *interface* to the exterior. More precisely, two networks are related with $\cong$ when, composed with the same third network, they offer the same *barbs* (i.e., the messages to external principals in the respective global queues are on the same channels), and this property is preserved under reduction.

We say that a message m is *routed for* α *in* N if $N=(\nu\tilde{n})(M_0 \mid \langle r\ ;\ h\rangle)$, $m\in h$, either $m=\overline{a}\langle s[\mathtt{r}]:T\rangle$ and $r(a)=\alpha$ or $m=s[\mathtt{r}_1,\mathtt{r}_2]!l\langle e\rangle$ and $r(s[\mathtt{r}_2])=\alpha$.

Definition 4.2 (Barb). We write $N\downarrow_a$ when the global queue of N contains a message m to free a, and m is routed for a principal not in N. We write $N\Downarrow_a$ if $N \longrightarrow^* N'\downarrow_a$.

We denote $\mathcal{P}(N)$ for the set of principals in N, that is $\mathcal{P}(\prod[P_i]_{\alpha_i})=\{\alpha_1,...,\alpha_n\}$. We say N_1 and N_2 are *composable* when $\mathcal{P}(N_1)\cap\mathcal{P}(N_2)=\emptyset$, the union of their routing tables remains a function, and their free session names are disjoint. If N_1 and N_2 are composable, we define $N_1\diamond N_2=(\nu\tilde{n}_1,\tilde{n}_2)(M_1 \mid M_2 \mid \langle r_1\cup r_2\ ;\ h_1\cdot h_2\rangle)$ where $N_i=(\nu\tilde{n}_i)(M_i \mid \langle r_i\ ;\ h_i\rangle)$ $(i=1,2)$.

Definition 4.3 (Barbed reduction-closed congruence). A relation $\mathcal{R}$ on networks with the same principals is a *barbed r.c. congruence* [32] if the following holds:

whenever $N_1 \mathcal{R} N_2$ we have: (1) for each composable N, $N \diamond N_1 \mathcal{R} N \diamond N_2$; (2) $N_1 \longrightarrow N_1'$ implies $N_2 \longrightarrow^* N_2'$ s.t. $N_1' \mathcal{R} N_2'$ again, and the symmetric case; (3) $N_1' \Downarrow_a$ iff $N_2' \Downarrow_a$. We write $N_1 \cong N_2$ when N_1 and N_2 are related by a barbed r.c. congruence.

Properties. The following result states that composing two bisimilar partial networks with the same network – implying the same router and global transport – yields two undistinguishable networks.

Proposition 4.4 (Congruency). If $M_1 \approx M_2$, then (1) $M_1|M \approx M_2|M$ for each composable partial network M; and (2) $M_1|N \cong M_2|N$ for each composable network N.

Proof. For (1) we show that the relation

$$\mathcal{R} = \{(M_1|M,\ M_2|M) \mid M_1 \approx M_2,\ M \text{ composable with } M_1 \text{ and } M_2\}$$

is a bisimulation. Suppose $(M_1|M)\mathcal{R}(M_2|M)$ and $M_1|M \xrightarrow{\ell} M^1$. We discuss the shape of M^1:

- If $M^1 = M_1'|M$, it means that $M_1 \xrightarrow{\ell} M_1'$. By definition of $\mathcal{R}$, $M_2 \xRightarrow{\hat{\ell}} M_2'$ and $M_1' \approx M_2'$, we conclude.
- If $M^1 = M_1|M'$, it means that $M \xrightarrow{\ell} M'$. It is easy to conclude.

By examining the reduction rule associated to parallel composition, we observe that no reduction is induced through interactions between the two networks. Hence we have covered all cases. The symmetric case (when $M_2|M \xrightarrow{\ell} M^2$) is similar.

To prove (2) we proceed by showing that

$$\mathcal{R} = \{((\nu\tilde{n})(M_1|N), (\nu\tilde{n})(M_2|N)) \mid M_1 \approx M_2,\ N \text{ composable with } M_1 \text{ and } M_2\}$$

is a barbed congruence. First, $\mathcal{R}$ is clearly a congruence since it is closed under composition. Second, for (2), we take a composable N'. We have $N' \diamond (M_i|N) = M_i|(N' \diamond N)$ for $i \in \{1, 2\}$. We use the definition of $\mathcal{R}$ to conclude. For (3), assume $M_1|N \longrightarrow N^1$.

- If $N^1 = M_1|N'$, it means that $N \longrightarrow N'$. We use the definition of $\mathcal{R}$ to conclude.
- If $N^1 = M_1'|N'$, it means that $N = M_0|\langle r\ ;\ \ell \cdot H\rangle$, $N = M_0'|\langle r\ ;\ H\rangle$ and $M_1 \xrightarrow{\ell} M_1'$. We deduce $N^2 = M_2'|N'$, with $N = M_0|\langle r\ ;\ \ell \cdot H\rangle$, $N = M_0'|\langle r\ ;\ H\rangle$ and $M_2 \xrightarrow{\ell} M_2'$. We use the definition of $\mathcal{R}$ to conclude.

- If the reduction is induced by interaction between M_1 and N, then M_2 has the corresponding action, hence we can reason in the same way, hence done.

For (2), we suppose that $(M_1|N) \Downarrow_\ell$. Two cases can occur:

- Either $N \Downarrow_\ell$ and it follows directly that $(M_2|N) \Downarrow_\ell$.
- or $M_1 \xrightarrow{\ell} M_1'$ and by definition of $\mathcal{R}$, $M_2 \xRightarrow{\ell} M_2'$, meaning that $(M_2|N) \Downarrow_a$.

The symmetric case is similar.

By definition this shows $\approx \subset \cong$.

Example 4.3 (Example of Equivalence). We give here an example illustrating our equivalences of networks. Consider the following networks:

$$
\begin{array}{lcl}
M_0' & = & [a_1(y_1[\mathtt{C}] : T_{\mathtt{C}}).y_1[\mathtt{C},\mathtt{A}]!\langle \mathtt{Login}(x_i)\rangle.y_1[\mathtt{A},\mathtt{C}]?\mathtt{LoginOK}().P_{\mathtt{LOOP},\mathtt{C}}]_{\alpha_1} \\
M_1' & = & [a_2(y_2[\mathtt{S}] : T_{\mathtt{S}}).P_{\mathtt{LOOP},\mathtt{S}}]_{\alpha_2} \\
 & & \mid [(\nu s)\ \overline{a_1}\langle s[\mathtt{C}] : T_{\mathtt{C}}\rangle \mid \overline{a_2}\langle s[\mathtt{S}] : T_{\mathtt{S}}\rangle \mid \overline{a_3}\langle s[\mathtt{A}] : T_{\mathtt{A}}\rangle \\
 & & \quad \mid a_3(y_3[\mathtt{A}] : T_{\mathtt{A}}).y_3[\mathtt{C},\mathtt{A}]?(\mathtt{Login}(x_i)).y_3[\mathtt{A},\mathtt{S}]!\langle \mathtt{LoginOk}()\rangle.P_{\mathtt{LOOP},\mathtt{A}}]_\beta \\
M_2' & = & [a_2(y_2[\mathtt{S}] : T_{\mathtt{S}}).P_{\mathtt{LOOP},\mathtt{S}}]_{\alpha_2} \\
 & & \mid [a_4(y_4[\mathtt{DB}] : T_{\mathtt{DB}}).y_4[\mathtt{A},\mathtt{DB}]?(\mathtt{Query}).y_4[\mathtt{DB},\mathtt{A}]!\langle \mathtt{Answer}\rangle]_\gamma \\
 & & \mid [(\nu s)\ (\overline{a_1}\langle s[\mathtt{C}] : T_{\mathtt{C}}\rangle \mid \overline{a_2}\langle s[\mathtt{S}] : T_{\mathtt{S}}\rangle \mid \overline{a_3}\langle s[\mathtt{A}] : T_{\mathtt{A}}\rangle \mid \overline{a_4}\langle s[\mathtt{DB}] : T_{\mathtt{DB}}\rangle) \\
 & & \quad \mid a_3(y_3[\mathtt{A}] : T_{\mathtt{A}}').y_3[\mathtt{C},\mathtt{A}]?(\mathtt{Login}(x_i)). \\
 & & \quad\ y_3[\mathtt{A},\mathtt{DB}]!\langle \mathtt{Query}\rangle.y_3[\mathtt{DB},\mathtt{A}]?(\mathtt{Answer}).P_{\mathtt{LOOP},\mathtt{A}}]_\beta \\
N_1' & = & M_1' \mid \langle a_1 \mapsto \alpha_1, a_2 \mapsto \alpha_2, a_3 \mapsto \beta\ ;\ \emptyset\rangle \\
N_2' & = & (\nu a_4)\ (M_2' \mid \langle a_1 \mapsto \alpha_1, a_2 \mapsto \alpha_2, a_3 \mapsto \beta, a_4 \mapsto \gamma\ ;\ \emptyset\rangle)
\end{array}
$$

Our networks implement the ATM example defined in 2.1. For the sake of clarity, we have to take the following shortcuts: (1) we only consider the login phase of the protocol, the LOOP phase is abstracted into three processes $P_{\mathtt{LOOP},\mathtt{C}}$, $P_{\mathtt{LOOP},\mathtt{A}}$, $P_{\mathtt{LOOP},\mathtt{S}}$ for the three different roles, (2) to lighten the notations, we do not make the logical annotations explicit, (3) as a result of (2), we do not implement login validation and only write the case were the login succeeds.

We present two different networks N_1' and N_2', both are implementing the Server-Authenticator part of the ATM protocol. The Server part is the same in both processes (executed at principal α_2), but the Authenticator part (executed at β) is different: N_1' implements straightforwardly the protocol while N_2' contains another indirection involving a fourth participant (executed at γ): the Authenticator sends a query to a *Database* to retrieve additional information required in the login process, and the Database answers.

Thus, the protocols implemented in both networks are different, as one involves three participants and the other one four. Yet, the query to the Database

is N_2' is unobservable from the outside, and an external client, such as N_0' cannot distinguish between N_1' and N_2'.

This is captured by our equivalences: the two partial networks M_1' and M_2' do not contain the same components, and as a result, are not bisimilar: after some steps, M_2' is able to emit on the channel a_4, which is impossible for M_1'. However, when encapsulated into dynamic networks N_1' and N_2', they are barbed r.c. congruent: they will offer it the same interface to the same external client.

Satisfaction. We finally present a satisfaction relation for partial networks that include local principals. If M is a partial network, $\models M \triangleright \Sigma$ s.t. $\mathsf{dom}(\Sigma) = \mathcal{P}(M)$, it means that: the specification Σ allows all the outputs from the network; the network M is ready to receive all the inputs indicated by the specification; and this is preserved by transition.

Definition 4.5 (Satisfaction). Let $\mathsf{sbj}(\ell)$ denote the subject of $\ell \neq \tau$. A relation $\mathcal{R}$ from partial networks to specifications is a *satisfaction* when $M\mathcal{R}\Sigma$ implies:

1. If $\Sigma \xrightarrow{\ell} \Sigma'$ for an input ℓ and M has an input at $\mathsf{sbj}(\ell)$, then $M \xrightarrow{\ell} M'$ s.t. $M'\mathcal{R}\Sigma'$.
2. If $M \xrightarrow{\ell} M'$ for an output at ℓ, then $\Sigma \xrightarrow{\ell} \Sigma'$ s.t. $M'\mathcal{R}\Sigma'$.
3. If $M \xrightarrow{\tau} M'$, then $\Sigma \xrightarrow{\tau} \Sigma'$ s.t. $M'\mathcal{R}\Sigma'$ (i.e. $M'\mathcal{R}\Sigma$ since $\Sigma \xrightarrow{\tau} \Sigma$ always).

When $M\mathcal{R}\Sigma$ for a satisfaction relation $\mathcal{R}$, we say M *satisfies* Σ, denoted $\models M \triangleright \Sigma$. By Definition 4.5 and Proposition 4.4 we obtain:

Proposition 4.6 (Satisfaction). If $M_1 \cong M_2$ and $\models M_1 \triangleright \Sigma$ then $\models M_2 \triangleright \Sigma$.

That is, if two networks present the same interface, they satisfy the same specifications.

5. Safety Assurance in Partial Networks

In this section, we present the properties underpinning safety assurance in partial networks, that are networks without a global transport. By considering partial networks we focus on the properties of principals (and their respective monitors) with respect to specifications, abstracting from the routing mechanisms. The routing mechanisms will be taken into account in later sections. We first consider networks consisting of single monitored principals (*local* safety) and then extend the results to partial networks in general (*global* safety).

Recall that: partial networks are networks without global transport; M denotes an unmonitored partial network; N denotes an unmonitored network; N denotes a (monitored or unmonitored) network. Monitors, ranged over by M, are specifications (of the form $\alpha : \langle \Gamma; \Delta \rangle$) used for dynamic verification. See Table 5 for a summary of the notation.

M	monitor	specification used for monitoring
M	unmonitored partial network	without M, without $\langle r \,;\, h\rangle$
N	unmonitored network	without M, with $\langle r \,;\, h\rangle$
N	network	with or without M, with $\langle r \,;\, h\rangle$

Table 1: Networks: summary of the notations

The partial network composed by a principal guarded by its monitor can take any action expected by the specification:

Lemma 5.1. For any principal $[P]_\alpha$, specification $\alpha : \langle \Gamma, \Delta\rangle$, and action ℓ, if $\alpha : \langle \Gamma, \Delta\rangle \xrightarrow{\ell} \alpha : \langle \Gamma', \Delta'\rangle$ and $[P]_\alpha \xrightarrow{\ell} [P']_\alpha$, then $[P]_\alpha \mid \alpha : \langle \Gamma, \Delta\rangle \xrightarrow{\ell} [P']_\alpha \mid \alpha : \langle \Gamma', \Delta'\rangle$.

Proof. Direct, as no interaction can appear between $[P]_\alpha$ and its monitor with specification $\alpha : \langle \Gamma, \Delta\rangle$ when ℓ is performed. □

Local safety ensures that a monitored process always behaves well with respect to the specification used to define its monitor.

Theorem 5.2 (Local safety). $\models [P]_\alpha \mid \mathsf{M} \triangleright \alpha : \langle \Gamma; \Delta\rangle$ with $\mathsf{M} = \alpha : \langle \Gamma; \Delta\rangle$.

Proof. We define a relation R as:

$$R = \{([P]_\alpha \mid \mathsf{M},\ \alpha : \langle \Gamma; \Delta\rangle) \ \mid\ \mathsf{M} = \alpha : \langle \Gamma; \Delta\rangle\}$$

Assume $([P_0]_{\alpha'} \mid \mathsf{M}_0, \alpha' : \langle \Gamma_0, \Delta_0\rangle) \in R$:

1. For an input ℓ, because $\mathsf{M}_0 = \alpha' : \langle \Gamma_0, \Delta_0\rangle$ by assumption, that $\alpha' : \langle \Gamma_0, \Delta_0\rangle \xrightarrow{\ell} \alpha' : \langle \Gamma'_0, \Delta'_0\rangle$ and $[P_0]_{\alpha'} \mid \mathsf{M}_0$ having an input at $\mathsf{sbj}(\ell)$ together imply that $[P_0]_{\alpha'} \xrightarrow{\ell} [P'_0]_{\alpha'}$, thus by Lemma 5.1, we have $[P_0]_{\alpha'} \mid \alpha' : \langle \Gamma_0, \Delta_0\rangle \xrightarrow{\ell} [P'_0]_{\alpha'} \mid \mathsf{M}'_0$, and $\mathsf{M}'_0 = \alpha' : \langle \Gamma'_0, \Delta'_0\rangle$. Thus we have $([P'_0]_{\alpha'} \mid \mathsf{M}'_0, \alpha' : \langle \Gamma'_0, \Delta'_0\rangle) \in R$.
2. For an output ℓ, $[P_0]_{\alpha'} \mid \mathsf{M}_0 \xrightarrow{\ell} [P'_0]_{\alpha'} \mid \mathsf{M}'_0$ implies $\mathsf{M}_0 = \alpha' : \langle \Gamma_0, \Delta_0\rangle \xrightarrow{\ell} \alpha' : \langle \Gamma'_0, \Delta'_0\rangle = \mathsf{M}'_0$. Thus we have $([P'_0]_{\alpha'} \mid \mathsf{M}'_0, \alpha' : \langle \Gamma'_0, \Delta'_0\rangle) \in R$.
3. For an τ, $[P_0]_{\alpha'} \mid \mathsf{M}_0 \xrightarrow{\tau} [P_0]_{\alpha'} \mid \mathsf{M}_0$ implies that $\mathsf{M}_0 = \alpha' : \langle \Gamma_0, \Delta_0\rangle \xrightarrow{\tau} \alpha' : \langle \Gamma_0, \Delta_0\rangle = \mathsf{M}_0$.

Therefore, by Definition 4.5, R is a satisfaction relation and $\models [P]_\alpha \mid \mathsf{M} \triangleright \alpha : \langle \Gamma; \Delta\rangle$ with $\mathsf{M} = \alpha : \langle \Gamma; \Delta\rangle$. □

We define a safety property for partial networks that may include multiple principals. It describes the fact that a monitored network satisfies its specification.

Definition 5.3 (Network global safety). $M \mid \mathsf{M}$ is globally safe with respect to Σ if and only if $\models M \mid \mathsf{M} \triangleright \Sigma$.

We introduce a condition on the structure of a network and on its monitors, which guarantees global safety. A partial network is *fully monitored w.r.t.* Σ when all its principals are monitored and the collection of the monitors is weakly bisimilar to Σ. Formally, $M \mid \mathsf{M}$ is fully monitored w.r.t. Σ when $M \mid \mathsf{M} \equiv [P_1]_{\alpha_1} \mid \mathsf{M}_1 \mid \ldots \mid [P_n]_{\alpha_n} \mid \mathsf{M}_n$ for some $n \geq 0$ and $\mathsf{M}_1, \ldots, \mathsf{M}_n \approx \Sigma$. By Theorem 5.4 a *fully monitored* network is globally safe. Theorem 5.4 justifies monitoring by ensuring that fully monitored systems behave as expected.

Theorem 5.4 (Global safety). If $M \mid \mathsf{M}$ is fully monitored w.r.t. Σ, then $\models M \mid \mathsf{M} \triangleright \Sigma$.

Proof. Assume N is composed by monitored endpoints $[P_i]_{\alpha_i} \mid \mathsf{M}_i, i \in \{1, ..., n\}$.

$$M \mid \mathsf{M} \equiv [P_1]_{\alpha_1} \mid \mathsf{M}_1 \mid ... \mid [P_n]_{\alpha_n} \mid \mathsf{M}_n$$

where $\mathsf{M}_i = \alpha_i : \langle \Gamma_i; \Delta_i \rangle$ for $i = \{1, ..., n\}$, $\Sigma = \mathsf{M}_1, ..., \mathsf{M}_n$. Based on Theorem 5.2, for each $i \in \{1, ..., n\}$,

$$\models [P_i]_{\alpha_i} \mid \mathsf{M}_i \triangleright \alpha_i : \langle \Gamma_i; \Delta_i \rangle$$

with $\mathsf{M}_i = \alpha_i : \langle \Gamma_i; \Delta_i \rangle$. By Definition 4.5 and induction, we have

$$[P_1]_{\alpha_1} \mid \mathsf{M}_1 \mid ... \mid [P_n]_{\alpha_n} \mid \mathsf{M}_n \triangleright \alpha_1 : \langle \Gamma_1; \Delta_1 \rangle, ..., \alpha_n : \langle \Gamma_n; \Delta_n \rangle$$

so that $\models M \mid \mathsf{M} \triangleright \Sigma$. □

6. Transparency of Monitored Networks

Whereas safety assurance focuses on preventing violations from the principals, transparency ensures that monitors do not affect the behaviour of well-behaved principals. We first consider transparency for partial networks consisting of one single principal (*local transparency*) and then extend the result to monitored networks with global transport.

Theorem 6.1 (Local transparency). If $\models [P]_\alpha \triangleright \alpha : \langle \Gamma; \Delta \rangle$, then $[P]_\alpha \approx ([P]_\alpha \mid \mathsf{M})$ with $\mathsf{M} = \alpha : \langle \Gamma; \Delta \rangle$.

That is, a correct participant is not impaired by monitoring.

Proof. Define a relation R as:

$$R = \{([P]_\alpha, [P]_\alpha \mid \mathsf{M}) \mid \models [P]_\alpha \triangleright \alpha : \langle \Gamma; \Delta \rangle\}$$

Assume $([P]_\alpha, [P]_\alpha \mid \mathsf{M}) \in R$,

- for an output ℓ (the case for τ is similar), $[P]_\alpha \xrightarrow{\ell} [P']_\alpha$ implies $\mathsf{M} \xrightarrow{\ell} \mathsf{M}'$ due to $\models [P]_\alpha \triangleright \mathsf{M}$; by Lemma 5.1, we have $[P]_\alpha \mid \mathsf{M} \xrightarrow{\ell} [P']_\alpha \mid \mathsf{M}'$;

- for an input ℓ, $[P]_\alpha \xrightarrow{\ell} [P']_\alpha$ only when $\mathsf{M} \xrightarrow{\ell} \mathsf{M}'$, which together imply that, by Lemma 5.1, $[P]_\alpha \mid \mathsf{M} \xrightarrow{\ell} [P']_\alpha \mid \mathsf{M}'$.

By Definition 4.5, we have $\models [P']_\alpha \triangleright \mathsf{M}'$, so that $([P']_\alpha, [P']_\alpha \mid \mathsf{M}') \in R$.

Symmetrically, since, by Theorem 5.2, we have $\models [P]_\alpha \mid \mathsf{M} \triangleright \alpha : \langle \Gamma; \Delta \rangle$ with $\mathsf{M} = \alpha : \langle \Gamma; \Delta \rangle$,

- for an output ℓ or τ, $[P]_\alpha \mid \mathsf{M} \xrightarrow{\ell} [P']_\alpha \mid \mathsf{M}'$ implies $\mathsf{M} \xrightarrow{\ell} \mathsf{M}'$ whenever $[P]_\alpha \xrightarrow{\ell} [P']_\alpha$;

- for an input ℓ, $[P]_\alpha \mid \mathsf{M} \xrightarrow{\ell} [P']_\alpha \mid \mathsf{M}'$ says $\mathsf{M} \xrightarrow{\ell} \mathsf{M}'$, which implies $[P]_\alpha \xrightarrow{\ell} [P']_\alpha$.

By Definition 4.5, we have $\models [P']_\alpha \mid \mathsf{M}' \triangleright \mathsf{M}'$, so that $([P']_\alpha \mid \mathsf{M}', [P']_\alpha) \in R$. By Definition 4.1, $[P]_\alpha \approx ([P]_\alpha \mid \mathsf{M})$ with $\mathsf{M} = \alpha : \langle \Gamma; \Delta \rangle$. □

By Proposition 4.4 and Theorem 6.1, we derive Corollary 6.2 stating that weakly bisimilar static networks combined with the same global transport are weakly bisimilar; i.e. monitoring does not affect routing of information to and from a correct principal.

Corollary 6.2 (Bisimilarity). If $\models [P]_\alpha \triangleright \alpha : \langle \Gamma; \Delta \rangle$, then for any $\langle r \ ; \ h \rangle$, we have $([P]_\alpha \mid \langle r \ ; \ h \rangle) \approx ([P]_\alpha \mid \mathsf{M} \mid \langle r \ ; \ h \rangle)$ with $\mathsf{M} = \alpha : \langle \Gamma; \Delta \rangle$.

Global transparency (Theorem 6.3) states a collection of specifications (monitors) does not alter the behaviour of a well-behaved networks. We consider networks *with global transport* to ensure that the correctness of the network is not altered during the routing of messages. Observe that the reduction relation for networks introduced in Fig. 5 models interactions with the global transport as *invisible* actions. In order to enable the observation of the behaviour of a network together with the dynamics of its global transport h we introduce a new set of rules for the labelled transitions of networks, denoted by $\xrightarrow{\ell}_{\mathrm{g}}$, and presented in Fig. 10. The transitions in Fig. 10 allow us to globally observe, for example, that a message sent by $[P]_\alpha$ *enters* the global transport:

$$\langle r \ ; \ h \rangle \xrightarrow{s[\mathtt{r}_1, \mathtt{r}_2]! l_j \langle v \rangle}_{\mathrm{g}} \langle r \ ; \ h \cdot s\langle \mathtt{r}_1, \mathtt{r}_2, l_j \langle v \rangle \rangle \rangle$$

$$\{\textsc{Req}\} \quad \langle r \,;\, h\rangle \xrightarrow{\overline{a}\langle s[\mathtt{r}]:T\rangle}_{\mathrm{g}} \langle r \,;\, h\cdot\overline{a}\langle s[\mathtt{r}]:T\rangle\rangle$$

$$\{\textsc{Acc}\} \quad \langle r \,;\, \overline{a}\langle s[\mathtt{r}]:T\rangle\cdot h\rangle \xrightarrow{a\langle s[\mathtt{r}]:T\rangle}_{\mathrm{g}} \langle r \,;\, h\rangle$$

$$\{\textsc{Sel}\} \quad \langle r \,;\, h\rangle \xrightarrow{s[\mathtt{r}_1,\mathtt{r}_2]!l\langle v\rangle}_{\mathrm{g}} \langle r \,;\, h\cdot\, s\langle \mathtt{r}_1,\mathtt{r}_2,l\langle v\rangle\rangle\rangle$$

$$\{\textsc{Bra}\} \quad \langle r \,;\, s\langle \mathtt{r}_1,\mathtt{r}_2,l\langle v\rangle\rangle\cdot h\rangle \xrightarrow{s[\mathtt{r}_1,\mathtt{r}_2]?l\langle v\rangle}_{\mathrm{g}} \langle r \,;\, h\rangle$$

$$\{\textsc{Net}\}\frac{N=[P]_\alpha \,|\, \langle r \,;\, h\rangle \quad [P]_\alpha \xrightarrow{\ell} [P']_\alpha \quad \langle r \,;\, h\rangle \xrightarrow{\ell}_{\mathrm{g}} \langle r' \,;\, h'\rangle \quad N'=[P']_\alpha \,|\, \langle r' \,;\, h'\rangle}{N \xrightarrow{\ell}_{\mathrm{g}} N'}$$

$$\{\textsc{tau}\}\ \frac{N \longrightarrow N'}{N \xrightarrow{\tau}_{\mathrm{g}} N'} \quad \{\textsc{res}\}\ \frac{N \xrightarrow{\ell}_{\mathrm{g}} N' \ a \notin \mathsf{sbj}(\ell)}{(\nu a)N \xrightarrow{\ell\backslash a}_{\mathrm{g}} (\nu a)N'} \quad \{\textsc{str}\}\frac{N \equiv N_0 \xrightarrow{\ell}_{\mathrm{g}} N_0' \equiv M'}{N \xrightarrow{\ell}_{\mathrm{g}} N'}$$

$$\{\textsc{par}\}\frac{N_1 \xrightarrow{\ell}_{\mathrm{g}} N_1' \quad \mathsf{bn}(\ell)\cap\mathsf{fn}(N_2)=\emptyset \quad \mathsf{dest}(\ell)\notin\mathcal{P}(N_2)}{N_1 \parallel N_2 \xrightarrow{\ell}_{\mathrm{g}} N_1' \mid N_2}$$

$$\{\textsc{Mon}\}\frac{\mathsf{N}=N\mid\mathsf{M} \quad N \xrightarrow{\ell}_{\mathrm{g}} N' \quad \mathsf{M}\xrightarrow{\ell}\mathsf{M}' \quad \mathsf{N}'=N'\mid\mathsf{M}'}{\mathsf{N}\xrightarrow{\ell}_{\mathrm{g}}\mathsf{N}'}$$

Figure 10: LTS for networks

Similarly, the parallel composition of a principal sending $s[\mathtt{r}_1,\mathtt{r}_2]!l_j\langle v\rangle$ and the global transport is made visible as follows:

$$[s[\mathtt{r}_1,\mathtt{r}_2]!l_j\langle v\rangle; P']_\alpha \mid \langle r \,;\, h\rangle \xrightarrow{s[\mathtt{r}_1,\mathtt{r}_2]!l_j\langle v\rangle}_{\mathrm{g}} [P']_\alpha \mid h\cdot s\langle \mathtt{r}_1,\mathtt{r}_2,l_j\langle v\rangle\rangle$$

We define dest as a partial function mapping a label, which representing an action, to its destination as:

$$\begin{array}{rcl}\mathsf{dest} & ::= & \overline{a}\langle s[\mathtt{r}]:T\rangle \mapsto a \mid a\langle s[\mathtt{r}]:T\rangle \mapsto a \\ & \mid & s[\mathtt{r}_1,\mathtt{r}_2]!l\langle v\rangle \mapsto s[\mathtt{r}_2] \mid s[\mathtt{r}_1,\mathtt{r}_2]?l\langle v\rangle \mapsto s[\mathtt{r}_2]\end{array}$$

The notation of global observable transition $\xrightarrow{\ell}_{\mathrm{g}}$, used to denote *globally observable action* ℓ, is defined by the rules in Fig. 10. Rules $\{\textsc{req}\}$ and $\{\textsc{acc}\}$ (resp. $\{\textsc{sel}\}$ and $\{\textsc{bra}\}$) are for inserting and removing invitation messages (resp. messages in established sessions) from the global transport. Rules $\{\textsc{acc}\}$ and $\{\textsc{bra}\}$ represent that, as a message leaves the global queue, there should be a local principal receiving it as an input. Similarly, rules $\{\textsc{req}\}$ and $\{\textsc{sel}\}$ represent that, as

a message enters the global queue, there should be a local principal outputting it to the queue. By $\{\text{NET}\}$ for unmonitored networks, as $N \xrightarrow{\ell}_{\text{g}} N'$, it means $\exists [P]_\alpha \in N$, $[P]_\alpha \xrightarrow{\ell} [P']_\alpha$ (i.e. locally visible) such that $\langle r \ ; \ h\rangle \xrightarrow{\ell}_{\text{g}} \langle r' \ ; \ h'\rangle$ (i.e. globally visible). Rule $\{\text{TAU}\}$ summarizes the reduction rules defined in Section 3. Rule $\{\text{RES}\}$ and $\{\text{STR}\}$ are standard. Rule $\{\text{PAR}\}$ says that, the bound names of action ℓ should not be any free name appearing in network N_2, and it should not be absorbed by any process in network N_2 (i.e. its destination is not in N_2). By rule $\{\text{MON}\}$ for monitored networks, $\mathsf{N} \xrightarrow{\ell}_{\text{g}} \mathsf{N}'$ means $\exists [P]_\alpha \mid \mathsf{M} \in \mathsf{N}$, $[P]_\alpha \xrightarrow{\ell} [P']_\alpha$ and $\mathsf{M} \xrightarrow{\ell} \mathsf{M}'$ (i.e. locally visible) such that $\langle r \ ; \ h\rangle \xrightarrow{\ell}_{\text{g}} \langle r' \ ; \ h'\rangle$ (i.e. globally visible).

Theorem 6.3 (Global transparency). Assume N and N have the same global transport $\langle r \ ; \ h\rangle$. If N is fully monitored w.r.t. Σ and $N = M \mid \langle r \ ; \ h\rangle$ is unmonitored but $\models M \triangleright \Sigma$, then we have $\mathsf{N} \approx N$.

Proof. Define a relation R:

$$R = \{\mathsf{N}, N \ \mid \ N = M \mid \langle r \ ; \ h\rangle \text{ and } \models M \triangleright \Sigma\}$$

We prove that R is a standard strong bisimilar relation over $\xrightarrow{\ell}_{\text{g}}$. Note that, $M \triangleright \Sigma$ means $\forall [P_i]_{\alpha_i} \in M$, we have $\alpha_i : \langle \Gamma_i ; \Delta_i\rangle \in \Sigma$ and $\models [P_i]_{\alpha_i} : \alpha_i : \langle \Gamma_i ; \Delta_i\rangle$.

1. As $\mathsf{N} \xrightarrow{\ell}_{\text{g}} \mathsf{N}'$, it implies $\exists [P_j]_{\alpha_j} \mid \mathsf{M}_j \in \mathsf{N}$, $[P_j]_{\alpha_j} \xrightarrow{\ell} [P'_j]_{\alpha_j}$ and $\mathsf{M}_j \xrightarrow{\ell} \mathsf{M}'_j$ such that $\langle r \ ; \ h\rangle \xrightarrow{\ell}_{\text{g}} \langle r' \ ; \ h'\rangle$, and other monitored processes in N are not affected. When ℓ is an input, by Definition 4.5, since $\models M \triangleright \Sigma$, we should have $[P_j]_{\alpha_j} \xrightarrow{\ell} [P'_j]_{\alpha_j}$; when ℓ is an output or a τ action, by Definition 4.5, the transition of $[P_j]_{\alpha_j} \xrightarrow{\ell} [P'_j]_{\alpha_j}$ is able to take place. Both cases lead to $M \xrightarrow{\ell} M'$ and $\langle r \ ; \ h\rangle \xrightarrow{\ell}_{\text{g}} \langle r' \ ; \ h'\rangle$ so that $N = M \mid \langle r \ ; \ h\rangle \xrightarrow{\ell}_{\text{g}} M' \mid \langle r' \ ; \ h'\rangle = N'$, and $\models [P']_{\alpha_j} : \alpha_j : \langle \Gamma'_j ; \Delta'_j\rangle$ by Definition 4.5. $\alpha_j : \langle \Gamma'_j ; \Delta'_j\rangle$ is the resulting new configuration of α_j in Σ. Other specifications $\{\alpha_i : \langle \Gamma_i ; \Delta_i\rangle\}_{i \in I \setminus \{j\}} \in \Sigma$ are not affected. Let $\Sigma' = \alpha_j : \langle \Gamma'_j ; \Delta'_j\rangle, \{\alpha_i : \langle \Gamma_i ; \Delta_i\rangle\}_{i \in I \setminus \{j\}}$. Therefore, for the resulting new network $N' = M' \mid \langle r' \ ; \ h'\rangle$, we have $\models M' \triangleright \Sigma'$. Thus we have $(\mathsf{N}', N') \in R$.
2. For the symmetric case, as $N \xrightarrow{\ell}_{\text{g}} N'$, it implies $\exists [P_j]_{\alpha_j} \in N$, $[P_j]_{\alpha_j} \xrightarrow{\ell} [P'_j]_{\alpha_j}$ such that $\langle r \ ; \ h\rangle \xrightarrow{\ell}_{\text{g}} \langle r' \ ; \ h'\rangle$ and other processes in N are not affected. Since $\models M \triangleright \Sigma$, without loss of generality, let $\mathsf{M}_j = \alpha_j : \langle \Gamma_j ; \Delta_j\rangle$, then we have, for any ℓ, $[P_j]_{\alpha_j} \mid \mathsf{M}_j \xrightarrow{\ell} [P'_j]_{\alpha_j} \mid \mathsf{M}'_j$, where $\mathsf{M}'_j = \alpha_j : \langle \Gamma'_j ; \Delta'_j\rangle$. It makes $\langle r \ ; \ h\rangle \xrightarrow{\ell}_{\text{g}} \langle r' \ ; \ h'\rangle$, so that $\mathsf{N} \xrightarrow{\ell}_{\text{g}} \mathsf{N}'$. Since N'

is a fully monitored network, its static part (i.e. the part when the global transport is taken off from N'), say $[P_i]_{\alpha_i} \mid \{\mathsf{M}_i\}_{i\in I}$ where $\{\mathsf{M}_i\}_{i\in I} = \alpha_j : \langle\Gamma'_j;\Delta'_j\rangle, \{\alpha_i : \langle\Gamma_i;\Delta_i\rangle\}_{i\in I\setminus\{j\}}$, $\models [P_i]_{\alpha_i} \mid \{M_i\}_{i\in I} : \Sigma'$ where $\Sigma' = \alpha_j : \langle\Gamma'_j;\Delta'_j\rangle, \{\alpha_i : \langle\Gamma_i;\Delta_i\rangle\}_{i\in I\setminus\{j\}}$. Thus we have $(N', \mathsf{N}') \in R$. □

By Theorems 5.4 and 6.3, we can *mix* unmonitored principals with monitored principals still obtaining global safety assurance:

Corollary 6.4 (Mixed Network). If $M \mid \mathsf{M}$ is fully monitored with respect to Σ, $\models M' \triangleright \Sigma$, and $\mathcal{P}(M) \cap \mathcal{P}(M') = \emptyset$, then $\models (M \mid \mathsf{M}) \mid M' \triangleright \Sigma$.

In the above corollary, untyped M is monitored by M which specifies Σ, while M' is unmonitored but statically checked to conform to Σ. The result shows that they can safely be composed.

7. Session Fidelity

The property of session fidelity says that, whenever all the principals in a static network conform to their specifications, then all of the derivatives of this static network conform to evolutions of the initial global specification.

Global Safety vs Session Fidelity. Recall that global safety (Definition 5.3) only ensures that in a network where principals are well-behaved with respect to their local types, all interactions conform to the collection of these local types. Session fidelity is a stronger property than global safety (Definition 5.3) as illustrated in Example 7.1.

Example 7.1. Consider a simple global type

$$G = \mathtt{r}_1 \to \mathtt{r}_2 : \{l_1(x_1)\{x_1 > 9\}.G_1, l_2(x_2)\{x_2 < 10\}.G_2\}$$

and processes P and Q implementing roles $\mathtt{r}_1$ and $\mathtt{r}_2$ in established session s

$$\begin{array}{rcl} P & = & s[\mathtt{r}_1, \mathtt{r}_2]!l_1\langle 10\rangle.P' \\ Q & = & s[\mathtt{r}_1, \mathtt{r}_2]?\{l_i(x_i).Q_i\}_{i\in\{1,2\}} \end{array}$$

Suppose that during runtime P sends out message $s\langle \mathtt{r}_1, \mathtt{r}_2, l_1\langle 10\rangle\rangle$ but, Q receives a message, perhaps revised by an attack, $s\langle \mathtt{r}_1, \mathtt{r}_2, l_2\langle 8\rangle\rangle$. These actions satisfy global safety since satisfy the specifications of P and Q, namely they are *locally* well-behaved. This scenario (i.e., the content of the message being modified between a send and a corresponding receive action) does not conform to the intended global protocol. We define a property, session fidelity, that rules out the scenario above (Definition 7.8) and prove (Theorem 7.13) that fully monitored networks

with global transport satisfy session fidelity. This is due to the fact that: (1) s is a private session ID that can be viewed only by the participants in the session, (2) all principals are guarded by monitors hence all messages reaching the global transport are valid, and (3) the global transport preserves the values of the messages it gets.

Configurations. We define session fidelity after giving the labelled transition relation for *configurations*, and a few auxiliary definitions.

Definition 7.2 (Configuration). A configuration is denoted by $\Phi = \Sigma; \langle r \ ; \ h \rangle$, where all messages corresponding to the actions guarded by Σ are in h.

A configuration guides the global behaviours in a network. By including the global queue, we let configuration capture the *global* behaviour in a network, which accounts also for the correct routing and dispatch of messages. Before giving the semantics of configurations, it will be useful to define when and how configurations can be composed. Let $\mathcal{P}(\Phi)$ be the set of principals involving in Φ.

Definition 7.3 (Parallel composition of configurations). Let $\Phi_1 = \Sigma_1; \langle r_1 \ ; \ h_1 \rangle$ and $\Phi_2 = \Sigma_2 \ ; \ \langle r_2 \ ; \ h_2 \rangle$ be configurations. We say that Φ_1 and Φ_2 are *composable* whenever $\mathcal{P}(\Phi_1) \cap \mathcal{P}(\Phi_2) = \emptyset$ and the union of their routing tables remains a function. If Φ_1 and Φ_2 are composable, then we define the composition of Φ_1 and Φ_2 as: $\Phi_1 \odot \Phi_2 = \Sigma_1, \Sigma_2 \ ; \ \langle r_1 \cup r_2 \ ; \ h_1 \cdot h_2 \rangle$.

The formal semantics of configurations is defined by the LTS in Fig. 11. The behaviour of each principal in a network is guided by the specification Σ, and is observed by the global transport $\langle r \ ; \ h \rangle$. Except rules [Acc] and [Par], all rules are straightforward from the LTS of specifications (defined in Section 4.1) and the one of dynamic networks (Fig. 10). We comment below on the interesting rules.

1. Rule [Acc] indicates that, only when the invitation has been (internally) accepted by a principal in the network, the routing information registers $s[\mathtt{r}] \mapsto \alpha$. When we observe the global transport (externally), we only observe that an invitation is moved out from the global queue (which implies that it has been accepted). However, we do not know *who* accepts it. Only Σ tells which principal accepts this invitation, so that we can register it in the routing information using α.
2. Rule [Par] says if Φ_1 and Φ_3 are composable (Definition 7.3), after Φ_1 becomes as Φ_2, they are still composable.

Our framework relies on two structural (well-formedness) properties on specifications: *consistency* and *coherence*. Consistent specifications are the ones corresponding to well-formed concrete systems (i.e., where the session initiation procedures are well-regulated, and where the active sessions correspond to projections of some well-formed global type).

$$\text{[Req]}\quad \frac{\Sigma \xrightarrow{\overline{a}\langle s[\mathtt{r}]:T\rangle} \Sigma'}{\Sigma\,;\langle r\ ;\ h\rangle \xrightarrow{\overline{a}\langle s[\mathtt{r}]:T\rangle}_{\mathrm{g}} \Sigma'\,;\langle r\ ;\ h\cdot\overline{a}\langle s[\mathtt{r}]:T\rangle\rangle}$$

$$\text{[Acc]}\quad \frac{\alpha:\langle\Gamma, a:\mathtt{I}(T[\mathtt{r}]);\Delta\rangle\in\Sigma \quad \Sigma \xrightarrow{a\langle s[\mathtt{r}]:T\rangle} \Sigma'}{\Sigma\,;\langle r\ ;\ \overline{a}\langle s[\mathtt{r}]:T\rangle\cdot h\rangle \xrightarrow{a\langle s[\mathtt{r}]:T\rangle}_{\mathrm{g}} \Sigma'\,;\langle r, s[\mathtt{r}]\mapsto\alpha\ ;\ h\rangle}$$

$$\text{[Sel]}\quad \frac{\Sigma \xrightarrow{s[\mathtt{r}_1,\mathtt{r}_2]!l\langle v\rangle} \Sigma'}{\Sigma\,;\langle r\ ;\ h\rangle \xrightarrow{s[\mathtt{r}_1,\mathtt{r}_2]!l\langle v\rangle}_{\mathrm{g}} \Sigma'\,;\langle r\ ;\ h\cdot s\langle \mathtt{r}_1,\mathtt{r}_2,l\langle v\rangle\rangle\rangle}$$

$$\text{[Bra]}\quad \frac{\Sigma \xrightarrow{s[\mathtt{r}_1,\mathtt{r}_2]?l\langle v\rangle} \Sigma'}{\Sigma\,;\langle r\ ;\ s\langle \mathtt{r}_1,\mathtt{r}_2,l\langle v\rangle\rangle\cdot h\rangle \xrightarrow{s[\mathtt{r}_1,\mathtt{r}_2]?l\langle v\rangle}_{\mathrm{g}} \Sigma'\,;\langle r\ ;\ h\rangle}$$

$$\text{[Par]}\quad \frac{\Phi_1 \xrightarrow{\ell}_{\mathrm{g}} \Phi_2}{\Phi_1\odot\Phi_3 \xrightarrow{\ell}_{\mathrm{g}} \Phi_2\odot\Phi_3} \qquad \text{[Tau]}\quad \frac{\Sigma \xrightarrow{\tau} \Sigma}{\Sigma;\langle r\ ;\ h\rangle \xrightarrow{\tau}_{\mathrm{g}} \Sigma;\langle r\ ;\ h\rangle}$$

Figure 11: Labelled transition relation for configurations

Definition 7.4 (Consistent and coherent specifications). $\Sigma = \{\alpha_i:\langle\Gamma_i;\Delta_i\rangle\}_{i\in I}$ is *consistent* when

1. there is one and only one i such that $\Gamma_i \vdash a:\mathtt{I}(T[\mathtt{r}])$, and
2. as long as $a:\mathtt{O}(T[\mathtt{r}])$ exists in some Γ_i, $\exists\Gamma_j$ such that $a:\mathtt{I}(T[\mathtt{r}])\in\Gamma_j$; and
3. for any s appearing in any Δ_j, if $\{s[\mathtt{r}_k]:T_k\}_{1\leq k\leq n}$ is a collection appeared in $\{\Delta_i\}_{i\in I}$, there exists well-formed G such that $\mathsf{roles}(G)=\{\mathtt{r}_1,..,\mathtt{r}_n\}$ and $G\upharpoonright \mathtt{r}_i = T_i$.

Two specifications Σ_1 and Σ_2 are *coherent* when their union is a consistent specification.

Next, we define *receivability*, *configurational consistency* and *conformance* for configurations, which are based on the LTS of configurations and dynamic networks. Receivability entices the ability for a message in transit to reach its destination.

Definition 7.5 (Receivable configuration). Receivability of a configuration $\Sigma;\langle r\ ;\ h\rangle$ is defined by the following induction:

1. If h is empty then $\Sigma;\langle r\ ;\ h\rangle$ is receivable.
2. If $h\equiv m\cdot h'$, then $\Sigma;\langle r\ ;\ h\rangle$ is receivable when we have $\Sigma;\langle r\ ;\ m\cdot h'\rangle \xrightarrow{\ell}_{\mathrm{g}} \Sigma';\langle r'\ ;\ h'\rangle$, where ℓ corresponding to m, and $\Sigma';\langle r'\ ;\ h'\rangle$ is receivable.

A configuration $\Sigma; \langle r \ ; \ h \rangle$ is *configurationally consistent* if all of its multi-step global input transition derivatives can be performed and the resulting specifications Σ is consistent (according to Definition 7.4).

Definition 7.6 (Configurational consistency). A configuration $\Phi = \Sigma; \langle r \ ; \ h \rangle$ is *configurationally consistent* whenever

1. h is empty and Σ is consistent, or
2. h is not empty, $\Sigma; \langle r \ ; \ h \rangle$ is receivable, and after receiving all messages in h with $\Sigma \xrightarrow{\ell_1 \ldots \ell_n} \Sigma'$ (by the LTS in Figure 7), where $\ell_i, i = \{1, ..., n\}$ are inputs and, $\forall m \in h$, $\exists \ell \in \ell_1 \ldots \ell_n$ such that ℓ corresponds to m, we have Σ' is consistent.

In other words, $\Sigma; \langle r \ ; \ h \rangle$ is configurationally consistent if, in each of its derivatives, all messages in the transport can be "received" by some monitors in Σ and, after absorbing all these messages, the resulting Σ' is still consistent. Conformance links networks and configurations.

Definition 7.7 (Conformance to a configuration). Assume a network $N \equiv M \mid \langle r \ ; \ h \rangle$ is given. We say that N conforms to $\Sigma; \langle r \ ; \ h \rangle$ when:

1. h is empty, $\models M \triangleright \Sigma$ and Σ is consistent, or
2. h is not empty, and the following conditions hold
   (a) $\models M \triangleright \Sigma$,
   (b) all messages in h are receivable to M, and
   (c) as $\Sigma; \langle r \ ; \ h \rangle \xrightarrow{\ell_1 \ldots \ell_n}_{\mathrm{g}} \Sigma'; \langle r' \ ; \ \emptyset \rangle$ so that $M \mid h \xrightarrow{\ell_1 \ldots \ell_n}_{\mathrm{g}} M' \mid \emptyset$ where each $\ell_i, i = \{1, ..., n\}$ is an input, Σ' is consistent.

Session Fidelity. Session fidelity describes the relation between a network and the configuration specifying it: all evolutions of the network should correspond to expected evolutions of the configuration which does not lead to ill-formed configurations. We now give the formal definition of session fidelity.

Definition 7.8 (Session fidelity). Assume configuration $\Sigma; \langle r \ ; \ h \rangle$ is configurationally consistent. We say that N satisfies session fidelity w.r.t. $\Sigma; \langle r \ ; \ h \rangle$ if and only if, for any ℓ, $N \xrightarrow{\ell}_{\mathrm{g}} N'$ implies $\Sigma; \langle r \ ; \ h \rangle \xrightarrow{\ell}_{\mathrm{g}} \Sigma'; \langle r' \ ; \ h' \rangle$ and $\Sigma'; \langle r' \ ; \ h' \rangle$ is configurationally consistent and N' satisfies session fidelity w.r.t. $\Sigma'; \langle r' \ ; \ h' \rangle$.

Before proving session fidelity for our monitored framework we give a few auxiliary lemmas. Lemma 7.9 states that, as a network conforms to some configurationally consistent configuration, the evolution of the configuration must be able to consume an output occurrence in the network:

Lemma 7.9. Assume a network $N \equiv M|\langle r\ ;\ h\rangle$ conforms to $\Sigma; \langle r\ ;\ h\rangle$, and that $\Sigma; \langle r\ ;\ h\rangle$ is configurationally consistent. If $N \xrightarrow{\ell}_{\mathrm{g}} N'$ with ℓ being an output and $\Sigma; \langle r\ ;\ h\rangle \xrightarrow{\ell}_{\mathrm{g}} \Sigma'; \langle r\ ;\ h{\cdot}m\rangle$, then $\Sigma'; \langle r\ ;\ h{\cdot}m\rangle$ is receivable.

Proof. We only show the interesting case. When $\ell = \overline{a}\langle s[\mathtt{r}]\ :\ T\rangle$, since Σ is consistent, by Definitions 7.4, there exists $a : \mathtt{I}(T[\mathtt{r}])$ in some Γ of Σ. Because ℓ does not affect the existence of $a : \mathtt{I}(T[\mathtt{r}])$, it remains in Γ of Σ', thus invitation $m = \overline{a}\langle s[\mathtt{r}] : T\rangle$ is receivable to Σ'.

Let $\alpha_i = \langle \Gamma_i, \Delta_i\rangle$. When $\ell = s[\mathtt{r}_1, \mathtt{r}_2]!l_j\langle v\rangle$, by Definitions 7.4 and 7.7, since $\models M \triangleright \Sigma$ and Σ is consistent, $\exists \alpha_s, \alpha_r \in \Sigma$, $\exists G$ is well-formed of the form

$$G = \mathtt{r}_1 \to \mathtt{r}_2 : \{l_i(x_i : (T[\mathtt{r}])_i)\{A_i\}.G_i\}_{i\in I}$$

such that s obeys to G:

$$\begin{array}{rcl} \Delta_s(s[\mathtt{r}_1]) & = & G \upharpoonright \mathtt{r}_1 = \mathtt{r}_2!\{l_i(x_i : (T[\mathtt{r}])_i)\{A_i\}.G_i \upharpoonright \mathtt{r}_1\}_{i\in I} \\ \Delta_r(s[\mathtt{r}_2]) & = & G \upharpoonright \mathtt{r}_2 = \mathtt{r}_1?\{l_i(x_i : (T[\mathtt{r}])_i)\{A'_i\}.G_i \upharpoonright \mathtt{r}_2\}_{i\in I} \end{array} \quad (1)$$

As action $s[\mathtt{r}_1, \mathtt{r}_2]!l_j\langle v\rangle$ fires, Equation 1 changes to

$$\begin{array}{rcl} \Delta_s(s[\mathtt{r}_1]) & = & G_j \upharpoonright \mathtt{r}_1 \\ \Delta_r(s[\mathtt{r}_2]) & = & G \upharpoonright \mathtt{r}_2 = \mathtt{r}_1?\{l_i(x_i : (T[\mathtt{r}])_i)\{A'_i\}.G_i \upharpoonright \mathtt{r}_2\}_{i\in I} \end{array}$$

the receiving capability of $\mathtt{r}_1?$ still remains in $\Delta_r(s[\mathtt{r}_2])$, where $\alpha_r \in \Sigma'$, thus $m = s\langle \mathtt{r}_1, \mathtt{r}_2, l_j\langle v\rangle\rangle$ is receivable to Σ'.

As $N \equiv M \mid H$ and $\models M \triangleright \Sigma$, the satisfaction relation of M and Σ remains whenever action takes place.

Lemma 7.10 says that, if the static part of a network satisfies a specification, then the evolution of the static part still satisfies the corresponding evolution of the specification.

Lemma 7.10. Assume $N \equiv M \mid H$ and $\models M \triangleright \Sigma$. If $N \xrightarrow{\ell}_{\mathrm{g}} N' \equiv M' \mid H'$ and $\Sigma \xrightarrow{\ell} \Sigma'$, then $\models M' \triangleright \Sigma'$.

Proof. Directly from Definition 4.5.

Finally, Lemma 7.12 states that, if a network conforms to a configurationally consistent configuration, then any evolution of the network conforms to the corresponding evolution of the configuration, which is still configurationally consistent. Lemma 7.12 relier on the definition of routing table given below.

Definition 7.11 (Routing table). . We define $\mathsf{route}(\Sigma)$, *the routing table derived from* Σ, as follows:

$$\begin{array}{l}\mathsf{route}(\alpha : \langle \Gamma; \Delta, s[\mathtt{r}] : T\rangle, \Sigma) = s[\mathtt{r}] \mapsto \alpha, \mathsf{route}(\alpha : \langle \Gamma; \Delta\rangle, \Sigma)\\ \mathsf{route}(\alpha : \langle \Gamma, a : \mathtt{I}(T[\mathtt{r}]); \Delta\rangle, \Sigma) = a \mapsto \alpha, \mathsf{route}(\alpha : \langle \Gamma; \Delta\rangle, \Sigma)\\ \mathsf{route}(\alpha : \langle \Gamma, a : \mathtt{O}(T[\mathtt{r}]); \Delta\rangle, \Sigma) = \mathsf{route}(\alpha : \langle \Gamma; \Delta\rangle, \Sigma)\end{array}$$

The routing table is used to observe inputs. Note that by Definition 7.4 (2), as long as Σ is consistent, the existence of $a : \mathtt{O}(T[\mathtt{r}])$ in Γ implies that the corresponding $a : \mathtt{I}(T[\mathtt{r}])$ is also in Γ.

Lemma 7.12. Assume configuration $\Sigma; \langle r \ ; \ h\rangle$ is configurationally consistent, and network $N \equiv M|\langle r \ ; \ h\rangle$ conforms to configuration $\Sigma; \langle r \ ; \ h\rangle$. Then for any ℓ, whenever we have $N \xrightarrow{\ell}_{\mathrm{g}} N'$ such that $\Sigma; \langle r \ ; \ h\rangle \xrightarrow{\ell}_{\mathrm{g}} \Sigma'; \langle r' \ ; \ h'\rangle$, it holds that $\Sigma'; \langle r' \ ; \ h'\rangle$ is configurationally consistent and that N' conforms to $\Sigma'; \langle r' \ ; \ h'\rangle$.

Proof. Assume N conforms to $\Sigma; \langle r \ ; \ h\rangle$, which is configurationally consistent. We prove the statement by inspection of each case.

(Sel) Let $\ell = s[\mathtt{r}_1, \mathtt{r}_2]!l_j\langle v\rangle$, $N \xrightarrow{\ell}_{\mathrm{g}} N'$ and $\Sigma; \langle \mathsf{route}(\Sigma) \ ; \ h\rangle \xrightarrow{\ell}_{\mathrm{g}} \Sigma'; \langle r \ ; \ h{\cdot}m\rangle$, where $m = s\langle \mathtt{r}_1, \mathtt{r}_2, l_j\langle v\rangle\rangle$.

Then $r = \mathsf{route}(\Sigma) = \mathsf{route}(\Sigma')$ because there is no change to the elements in Σ or to the routing table.

Since Σ allows ℓ and Σ is consistent, then $\exists \alpha_r, \alpha_s \in \Sigma$, and $\exists G$ well-formed of the form

$$G = \mathtt{r}_1 \to \mathtt{r}_2\{l_i(x_i : S_i)\{A_i\}.G_i\}_{i\in I},$$

such that

$$\begin{array}{rcl}\Delta_s(s[\mathtt{r}_1]) & = & G \upharpoonright \mathtt{r}_1 = \mathtt{r}_2!\{l_i(x_i : S_i)\{A_i\}.G_i \upharpoonright \mathtt{r}_1\}_{i\in I},\\ \Delta_r(s[\mathtt{r}_2]) & = & G \upharpoonright \mathtt{r}_2 = \mathtt{r}_1?\{l_i(x_i : S_i)\{A'_i\}.G_i \upharpoonright \mathtt{r}_2\}_{i\in I}.\end{array}$$

$\Sigma \xrightarrow{\ell} \Sigma'$ implies Σ' has

$$\begin{array}{rcl}\Delta_s(s[\mathtt{r}_1]) & = & G_j \upharpoonright \mathtt{r}_1,\\ \Delta_r(s[\mathtt{r}_2]) & = & \mathtt{r}_1?\{l_i(x_i : S_i)\{A'_i\}.G_i \upharpoonright \mathtt{r}_2\}_{i\in I}.\end{array}$$

Case 1: h is empty. By Lemma 7.9, after receiving m, say $\Sigma' \xrightarrow{\ell} \Sigma''$, Σ'' has $s[\mathtt{r}_1] = G_j \upharpoonright \mathtt{r}_1$ and $s[\mathtt{r}_2] = G_j \upharpoonright \mathtt{r}_2$, Σ'' is thus consistent by Definition 7.4. By Definition 7.6, $\Sigma'; \langle r \ ; \ m\rangle$ is configurationally consistent, and

$\models M' \triangleright \Sigma'$ by Lemma 7.10, thus N' conforms to $\Sigma'; \langle r \ ; \ h \cdot m \rangle$.

Case 2: h is not empty. Since $\Sigma; \langle r \ ; \ h \rangle$ is configurationally consistent, again, by Lemma 7.9, after receiving messages in h (but not m), say $\Sigma' \xrightarrow{\ell_0 \ldots \ell_n} \Sigma'_1$, where every action in $\ell_0 \ldots \ell_n$ corresponds to each message in h, we have $\Sigma'_1; \langle r' \ ; \ m \rangle$ is configurationally consistent. After Σ'_1 receives m, say $\Sigma'_1 \xrightarrow{s[\mathtt{p}_1, \mathtt{p}_2]?l\langle v \rangle} \Sigma''$, where $s[\mathtt{p}_1, \mathtt{p}_2]?l\langle v \rangle$ is dual to ℓ, with the same reasoning above, Σ'' has $s[\mathtt{r}_1] = G'_j \upharpoonright \mathtt{r}_1$ and $s[\mathtt{r}_2] = G'_j \upharpoonright \mathtt{r}_2$, so that Σ'' is consistent. By Definition 7.6, $\Sigma'; \langle r \ ; \ h \cdot m \rangle$ is configurationally consistent, and $\models M' \triangleright \Sigma'$ by Lemma 7.10, thus N' conforms to $\Sigma'; \langle r \ ; \ h \cdot m \rangle$.

(Bra) Let $\ell = s[\mathtt{r}_1, \mathtt{r}_2]?l_j\langle v \rangle$, $N \xrightarrow{\ell}_{\mathrm{g}} N'$ and N conforms to $\Sigma; \langle \mathsf{route}(\Sigma) \ ; \ h \rangle$.

Case 1: h is empty. Since $\Sigma; \langle \mathsf{route}(\Sigma) \ ; \ \emptyset \rangle \not\xrightarrow{\ell}_{\mathrm{g}}$, so this case never happens.

Case 2: h is not empty. Thus, $N \xrightarrow{\ell}_{\mathrm{g}} N'$ and

$$\Sigma; \langle \mathsf{route}(\Sigma) \ ; \ h \rangle \xrightarrow{\ell}_{\mathrm{g}} \Sigma'; \langle r \ ; \ h/m \rangle,$$

where h/m means taking off message m from h, where $m = s\langle \mathtt{r}_1, \mathtt{r}_2, l_j\langle v \rangle \rangle$

We have $r = \mathsf{route}(\Sigma) = \mathsf{route}(\Sigma')$ because there is no change to the elements in Σ or to the routing table. By Definition 7.6, after receiving all messages in H, Σ is consistent, thus Σ', which has received message m is consistent after receiving all messages in h/m. By Lemma 7.10, we have $\models M' \triangleright \Sigma'$ thus N' conforms to $\Sigma'; \langle r \ ; \ h/m \rangle$.

(Req) Let $\ell = \overline{a}\langle s[\mathtt{r}] : T \rangle$. $N \xrightarrow{\ell}_{\mathrm{g}} N'$ and

$$\Sigma; \langle \mathsf{route}(\Sigma) \ ; \ h \rangle \xrightarrow{\ell}_{\mathrm{g}} \Sigma'; \langle r \ ; \ h \cdot m \rangle,$$

where $m = \overline{a}\langle s[\mathtt{r}] : T \rangle$. Then $r = \mathsf{route}(\Sigma) = \mathsf{route}(\Sigma')$ because, by Definition 7.11, nothing new is registered to the routing table.

Since Σ allows ℓ and Σ is consistent, by Definition 7.4, $\exists \Gamma_i, \Gamma_j \in \Sigma$ such that $a : \mathtt{I}(T[\mathtt{r}]) \in \Gamma_i$ and $a : \mathtt{O}(T[\mathtt{r}]) \in \Gamma_j$. After $\Sigma \xrightarrow{\ell} \Sigma'$, by rule [REQ] in the LTS of specifications, $a : \mathtt{I}(T[\mathtt{r}])$ remains in Γ'_i, $a : \mathtt{O}(T[\mathtt{r}])$ remains in

Γ'_j, and thus they both remain in Σ'.

Case 1: h is empty. By Lemma 7.9, after receiving m, say $\Sigma' \xrightarrow{a\langle s[\mathtt{r}]:T\rangle} \Sigma''$, both $a : \mathtt{I}(T[\mathtt{r}])$ and $a : \mathtt{O}(T[\mathtt{r}])$ remain in Σ'', satisfying Definition 7.4, so that $\Sigma'; \langle r \ ; \ m\rangle$ is configurationally consistent. By Lemma 7.10, we have $\models M' \triangleright \Sigma'$, thus N' conforms to $\Sigma'; \langle r \ ; \ h{\cdot}m\rangle$.

Case 2: h is not empty. The proof is similar to the one in (**Sel**) and omitted.

(Acc) Let $\ell = a\langle s[\mathtt{r}] : T\rangle$.

Case 1: h is empty. Since $\Sigma; \langle \mathsf{route}(\Sigma) \ ; \ \emptyset\rangle \not\xrightarrow{\ell}_{\mathrm{g}}$, this case never happens.

Case 2: h is not empty. If $N \xrightarrow{\ell}_{\mathrm{g}} N'$ and

$$\Sigma; \langle \mathsf{route}(\Sigma) \ ; \ h\rangle \xrightarrow{\ell}_{\mathrm{g}} \Sigma'; \langle r' \ ; \ h/m\rangle,$$

where $m = \overline{a}\langle s[\mathtt{r}] : T\rangle$. Since there exists $\Delta \in \Sigma'$ s.t. $s[\mathtt{r}] \in \Delta$, by Definition 7.11, $r' = \mathsf{route}(\Sigma), s[\mathtt{r}] \mapsto \alpha = \mathsf{route}(\Sigma')$.

For the same reasoning in (Bra), we have $\Sigma'; \langle r \ ; \ h/m\rangle$ is configurationally consistent. By Lemma 7.10, we have $\models M' \triangleright \Sigma'$ thus N' conforms to $\Sigma'; \langle r \ ; \ h/m\rangle$.

The proof for other cases is trivial.

Theorem 7.13 (Session fidelity). If N is fully monitored and conforms to $\Sigma; \langle r \ ; \ h\rangle$, which is configurationally consistent, then N satisifies session fidelity.

Proof. The proof is straightforward by Lemma 7.12 and Definition 7.8.

Proposition 7.14. Whenever a network is fully monitored, global safety implies session fidelity.

Proof. Simply by Definitions 7.6 and 7.7 and Corollary 6.2 and Theorems 5.4 and 7.12.

8. Related Work

Monitors. Our work features a located, distributed process calculus to model *dynamic* monitored networks. An account of the state of the art of runtime monitors can be found in [29, 38]. According to Havelund and Goldberg [29], specification-based runtime verification consists of monitoring a program's execution against a user-provided specification of the intended program's behaviour. Leucker and Schallhart [38] define runtime verification as the discipline of dealing with the detection of violations (or satisfaction) of correctness properties. They point out the use of runtime verification for contract enforcement.

Global specification languages. Message Sequence Charts (MSC), which are also known as UML sequence diagrams, have been the focus of many works [29, 36, 27, 37]. Among them, Kruger et al. [37] propose a runtime monitoring framework based on projecting MSC to distributed monitors based on finite state machines. They use aspect-oriented programming techniques to inject the monitors into the implementation of the components. Gan [26] follows the same path, but with a centralised approach. Both works do not provide a formal model, formal guarantees of correctness, nor support behavioural analysis. BPEL [4, 5, 27] is an orchestration description language that is now a common part of many industrial distributed systems where web services must be used in a coordinated manner. It supports the definition of abstract specifications as well as their execution. BPEL specifications are designed to be run in a centralised way. Baresi et. al [4] developed a run-time monitoring tool with assertions based on BPEL as an execution language. When the execution of a BPEL process reaches the point where an assertion must be checked, the tool calls an external service to check its satisfaction. This work does not consider properties, such as transparency and local/-global safety. On another line of research, van der Aalst et al. [50] use abstract BPEL process as specifications. Their work focuses on checking conformance between *execution logs* (obtained by observing a number of executions based on SOAP message exchanges, and then translated into Petri Nets) and choreographies expressed as abstract BPEL processes. The focus of [50] is on checking conformance a posteriori, as well as on revealing (mining) and re-engineering choreographies according to the actual system's behaviour. Differently from [50] our work establishes a theory of dynamic monitoring. The aim of our work is to observe communication as they occur to prevent unsafe interactions, while providing a formal framework that complements static (behavioural) typing techniques, and supports reasoning about equivalence of networks. Finally, WS-CDL is a more recent description language which aims at describing decentralised choreographies. Cambronero et al. [11] transform choreographies written in WS-CDL into timed-automata and verify systems against them. The work in [11] does not

develop a projection algorithm nor ensures global conformance with respect to a choreography.

Theory of monitored networks. The work of Ferrari et al. [25] proposes an ambient-based run-time monitoring formalism, called guardians, targeted at access control rights for network processes, and Klaim [19] advocates a hybrid (dynamic and static) approach for access control against capabilities (policies) to support static checking integrated within a dynamic access-control procedure. These works address specific forms of access control for mobility, while our more general approach aims at ensuring correct behaviour in sessions through a combination of static and run-time verification.

The work of Capecchi et al. [12] presents a monitor-based information-flow analysis of multiparty sessions. The monitors in [12] are inline (following [14]) and control the information-flow by tagging each message with security levels. Since each inlined monitor is located within a local process, the interactions between endpoint processes and their corresponding monitors are *synchronous*. We study *asynchronous* communications that, while being closer to an actual network implementation, introduce considerable challenges in the development of a theory. Other works on inlined monitors, such as [28, 1, 49], provide a policy specification language. The aim is to write policies into the monitors, with the guarantee that the specifications in the inlined monitors satisfy the original policies. Inline monitors require direct access to the code, whereas our approach, outline monitoring (i.e., the implantation of monitors is independent from the implementation of the observed applications), ensures interoperability with any language and architecture. Other related works on monitoring conversations are [48, 2]. Simmonds et al. [48] propose a runtime monitoring approach based on MSC as a specification language to represent global protocols, and transform MSC specifications into automata. They provide conformance checking of *finite* execution traces against specifications. Ancona et al. [2, 40] propose a dynamic monitoring framework based on MPST for Multi-Agent Systems (MAS) to guard interactions between local agents and their environments. They gave a procedure that automatically derives a self-monitoring MAS from Jason (a MAS development platform), and verifies that a MAS implementation is compliant with a given *global session type*, which can naturally be represented as cyclic Prolog terms. Their monitoring is only synchronous. Their development focuses on implementation and does not involve proofs of formal properties.

Monitoring and MPST. An informal approach to monitoring based on MPST, and an outline of monitors are presented in [17]. However, [17] only gives an overview of the desired properties, and requires all local processes to be dynamically verified through the protections of system monitors. In this article, instead,

we integrate statically and dynamically verified local processes into one network, and formally state the properties of this combination. Some recent works [3, 18] use multiparty session types for dynamic updates. Anderson et al. [3] study channel conditions of running processes to be able to update them and ensure deadlock-freedom, while a system in Coppo et al. [18] enables to update global types dynamically. The work [18] is based on the formulation (without assertions) studied in this article. Recently, Jia and al. [35] proposed a linear-logic based session-calculus close to ours describing monitor semantics for higher-order sessions which include rules for *blame assignment*.

In summary, compared to these related works, our contribution focuses on the enforcement of global safety, with protocols specified as multiparty session types with assertions. It also provides formalisms and theorems for decentralised runtime monitoring, targeting interaction between components written in multiple (e.g., statically and dynamically typed) programming languages.

9. Conclusion and future work

We proposed a new formal safety assurance framework to specify and enforce global safety of distributed systems through dynamic verification. We formally proved the correctness of our architectural framework through a π-calculus based theory, identified in two key properties of dynamic networks: global transparency and safety. We introduced a behavioural theory over monitored networks which allows compositional reasoning over trusted and untrusted (but monitored) components.

Implementations. As a part of our collaboration with the Ocean Observatories Initiative [44], our theoretical framework is currently realised by an implementation [34, 43, 21], in which each monitor supports all well-formed protocols and is automatically self-configured, via session initiation messages, for all sessions that the endpoint participates in. Our implementation of the framework automates distributed monitoring by generating FSM from the local protocol projections. In this implementation, the global protocol serves as the key abstraction that helps unify the aspects of specification, implementation and verification (both static and dynamic) of distributed application development. Our experience has shown that the specification framework can accommodate diverse practical use cases, including real-world communication patterns used in the distributed services of the OOI cyberinfrastructure [44].

Future Work. Our objectives include the incorporation in the implementation of more elaborate handling of error cases into monitor functionality, such as halting all local sessions or coercing to valid actions [46, 39]. In order to reach this goal,

we need to combine a simplification of [13] and nested sessions [20] to handle exceptions inside MPST. We aim to construct a simple and reliable way to raise and catch exceptions in asynchronous networks. Another direct extension of this work would be the addition of states (memories) to the syntax, as described in [8, 15]. It would require the monitors to maintain a model of the state of the applications being monitored, which can be easily formalised in our setting. For the sake of clarity, we did not add to our local type syntax other syntactical constructs such as parallel composition but such an extension is possible and could be considered, as it allows one to reach greater expressiveness [22]. Our work is motivated by ongoing collaborations with the Savara[1] and Scribble[2] projects [51, 31] and OOI [44]. We are continuing the development of Scribble, its toolsuite and associated environments towards an integration into [44]. The theoretical framework developed in this article is extensible as a basis for other applications as demonstrated in our recent dynamic monitoring implementations for distributed actors [42] and timers [41]. For instance, the work in [41] extends run-time monitoring to real-time processes: monitors verify the punctuality of interactions against time constraints expressed as a timed extension of Scribble based on timed MPST [10].

References

[1] I. Aktug, M. Dam, and D. Gurov. Provably correct runtime monitoring. *J. Log. Algebr. Program.*, 78(5):304–339, 2009.

[2] D. Ancona, S. Drossopoulou, and V. Mascardi. Automatic generation of self-monitoring MASs from multiparty global session types in Jason. In *DALT*, volume 7784 of *LNCS*, pages 1–17. Springer, 2012.

[3] G. Anderson and J. Rathke. Dynamic software update for message passing programs. In *APLAS*, volume 7705 of *LNCS*, pages 207–222. Springer, 2012.

[4] L. Baresi, C. Ghezzi, and S. Guinea. Smart monitors for composed services. In *ICSOC*, pages 193–202. ACM, 2004.

[5] L. Baresi, S. Guinea, M. Pistore, and M. Trainotti. Dynamo + astro: An integrated approach for BPEL monitoring. *2009 IEEE International Conference on Web Services*, pages 230–237, 2009.

[6] L. Bettini, M. Coppo, L. D'Antoni, M. D. Luca, M. Dezani-Ciancaglini, and N. Yoshida. Global progress in dynamically interleaved multiparty sessions. In *CONCUR*, volume 5201 of *LNCS*, pages 418–433. Springer, 2008.

[1] `http://www.jboss.org/savara`
[2] `http://scribble.github.io/`

[7] L. Bocchi, T.-C. Chen, R. Demangeon, K. Honda, and N. Yoshida. Monitoring networks through multiparty session types. In *FMOODS/FORTE*, volume 7892 of *LNCS*, pages 50–65. Springer, 2013.

[8] L. Bocchi, R. Demangeon, and N. Yoshida. A multiparty multi-session logic. In *TGC*, volume 8191 of *LNCS*, pages 97–111. Springer, 2012.

[9] L. Bocchi, K. Honda, E. Tuosto, and N. Yoshida. A theory of design-by-contract for distributed multiparty interactions. In *CONCUR*, volume 6269 of *LNCS*, pages 162–176. Springer, 2010.

[10] L. Bocchi, W. Yang, and N. Yoshida. Timed multiparty session types. In *CONCUR*, volume 8704 of *LNCS*, pages 419–434. Springer, 2014.

[11] M.-E. Cambronero et al. Validation and verification of web services choreographies by using timed automata. *J. Log. Algebr. Program.*, 80(1):25–49, 2011.

[12] S. Capecchi, I. Castellani, and M. Dezani-Ciancaglini. Information flow safety in multiparty sessions. In *EXPRESS*, volume 64 of *EPTCS*, pages 16–30, 2011.

[13] S. Capecchi, E. Giachino, and N. Yoshida. Global escape in multiparty session. In *FSTTCS*, volume 8 of *LIPICS*, pages 338–351. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[14] F. Chen and G. Rosu. MOP: an efficient and generic runtime verification framework. In *OOPSLA*, pages 569–588. ACM, 2007.

[15] T. Chen and K. Honda. Specifying stateful asynchronous properties for distributed programs. In *CONCUR*, volume 7454 of *LNCS*, pages 209–224. Springer, 2012.

[16] T.-C. Chen. *Theories for Session-based Governance for Large-Scale Distributed Systems*. PhD thesis, Queen Mary, University of London, 2013.

[17] T.-C. Chen, L. Bocchi, P.-M. Deniélou, K. Honda, and N. Yoshida. Asynchronous distributed monitoring for multiparty session enforcement. In *TGC*, volume 7173 of *LNCS*, pages 25–45. Springer, 2011.

[18] M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Self-adaptive monitors for multiparty sessions. In *PDP*, pages 688–696. IEEE, 2014.

[19] R. De Nicola, G. Ferrari, and R. Pugliese. Klaim: a kernel language for agents interaction and mobility. *IEEE Trans. Softw. Eng.*, 24:315–330, 1998.

[20] R. Demangeon and K. Honda. Nested protocols in session types. In *CONCUR*, volume 7454 of *LNCS*, pages 272–286. Springer, 2012.

[21] R. Demangeon, K. Honda, R. Hu, R. Neykova, and N. Yoshida. Practical interruptible conversations: Distributed dynamic verification with multiparty session types and python. *FMSD*, pages 1–29, 2015.

[22] R. Demangeon and N. Yoshida. On the expressiveness of multiparty sessions. In *FSTTCS*, volume 45 of *LIPIcs*, pages 560–574. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[23] P.-M. Deniélou and N. Yoshida. Dynamic multirole session types. In *POPL*, pages 435–446. ACM, 2011.

[24] P.-M. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In *ESOP*, volume 7211 of *LNCS*, pages 194–213. Springer, 2012.

[25] G. L. Ferrari, E. Moggi, and R. Pugliese. Guardians for ambient-based monitoring. *Electr. Notes Theor. Comput. Sci.*, 66(3):52–75, 2002.

[26] Y. Gan, M. Chechik, S. Nejati, J. Bennett, B. O'Farrell, and J. Waterhouse. Runtime monitoring of web service conversations. In *CASCON*, pages 42–57. ACM, 2007.

[27] C. Ghezzi and S. Guinea. Run-time monitoring in service-oriented architectures. In *Test and Analysis of Web Services*, pages 237–264. Springer, 2007.

[28] K. W. Hamlen and M. Jones. Aspect-oriented in-lined reference monitors. In *PLAS*, pages 11–20. ACM, 2008.

[29] K. Havelund and A. Goldberg. Verify your runs. In *Verified Software: Theories, Tools, Experiments*, pages 374–383. Springer, 2008.

[30] K. Honda, R. Hu, R. Neykova, T.-C. Chen, R. Demangeon, P.-M. Denilou, and N. Yoshida. Structuring communication with session types. In *COB*, volume 8665 of *LNCS*, pages 105–127. Springer, 2012.

[31] K. Honda, A. Mukhamedov, G. Brown, T.-C. Chen, and N. Yoshida. Scribbling interactions with a formal foundation. In *ICDCIT*, volume 6536 of *LNCS*, pages 55–75. Springer, 2011.

[32] K. Honda and N. Yoshida. On reduction-based process semantics. *Theor. Comput. Sci.*, 151(2):437–486, 1995.

[33] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL*, pages 273–284. ACM, 2008.

[34] R. Hu, R. Neykova, N. Yoshida, and R. Demangeon. Practical interruptible conversations: distributed dynamic verication with session types and python. In *RV*, volume 8174 of *LNCS*, pages 148–130. Springer, 2013.

[35] L. Jia, H. Gommerstadt, and F. Pfenning. Monitors and blame assignment for higher-order session types. In *POPL*, pages 582–594. ACM, 2016.

[36] I. H. Krüger, M. Meisinger, and M. Menarini. Runtime verification of interactions: from MSCs to aspects. In *RV*, volume 4839 of *LNCS*, pages 63–74. Springer, 2007.

[37] I. H. Krüger, M. Meisinger, and M. Menarini. Interaction-based runtime verification for systems of systems integration. *J. Log. Comput.*, 20(3):725–742, 2010.

[38] M. Leucker and C. Schallhart. A brief account of runtime verification. *J. Log. Algebr. Program.*, 78(5):293–303, 2009.

[39] J. Ligatti, L. Bauer, and D. Walker. Run-time enforcement of nonsafety policies. *ACM Trans. Inf. Syst. Secur.*, 12:19:1–19:41, 2009.

[40] V. Mascardi and D. Ancona. Attribute global types for dynamic checking of protocols in logic-based multiagent systems. *TPLP*, 13(4-5-Online-Supplement), 2013.

[41] R. Neykova, L. Bocchi, and N. Yoshida. Timed runtime monitoring for multiparty conversations. In *BEAT*, volume 162 of *EPTCS*, pages 19–26, 2014.

[42] R. Neykova and N. Yoshida. Multiparty session actors. In *COORDINATION*, volume 8459 of *LNCS*, pages 131–146. Springer, 2014. A full version in *LMCS*.

[43] R. Neykova, N. Yoshida, and R. Hu. SPY: Local Verification of Global Protocols. In *RV*, volume 8174 of *LNCS*, pages 363–358. Springer, 2013.

[44] OOI. `http://www.oceanobservatories.org/`.

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

[46] F. B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3:30–50, 2000.

[47] Scribble Project homepage. http://www.scribble.org.

[48] J. Simmonds, Y. Gan, M. Chechik, S. Nejati, B. O'Farrell, E. Litani, and J. Waterhouse. Runtime monitoring of web service conversations. *IEEE T. Services Computing*, 2(3):223–244, 2009.

[49] M. Sridhar and K. W. Hamlen. Model-checking in-lined reference monitors. In *VMCAI*, volume 5944 of *LNCS*, pages 312–327. Springer, 2010.

[50] W. M. P. van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and E. Verbeek. Conformance checking of service behavior. *ACM Trans. Internet Techn.*, 8(3), 2008.

[51] N. Yoshida, R. Hu, R. Neykova, and N. Ng. The scribble protocol language. In *TGC*, volume 8358 of *LNCS*, pages 22–41. Springer, 2013.